

Appendix B1

```
/*
 * Lexer.
 * Copyright (c) 1998-1999 New Generation Software (NGS) Oy
 *
 * Author: Markku Rossi <mtr@ngs.fi>
 */

/*
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public
 * License along with this library; if not, write to the Free
 * Software Foundation, Inc., 59 Temple Place - Suite 330, Boston,
 * MA 02111-1307, USA.
 */

/*
 * The GNU Library General Public License may also be downloaded at
 * http://www.gnu.org/copyleft/gpl.html.
 */

/*****
 *
 * This software was modified by Yahoo! Inc. under the terms
 * of the GNU Library General Public License (LGPL). For all
 * legal, copyright, and technical issues relating to how
 * this software can be used under GNU LGPL, please write to:
 *
 * GNU Compliance, Legal Dept., Yahoo! Inc.,
 * 3420 Central Expressway, Santa Clara, California U.S.A.
 *
 *****/

var rjs_VTAB = '\013';      // @@ For IE

/*
 * $Source: /usr/local/cvsroot/ngs/js/jsc/lexer.js,v $
 * $Id: lexer.js,v 1.9 1999/01/11 08:56:30 mtr Exp $
 */

/*
 * Global functions.
 */

function JSC$lexer (stream)
{
```

00650273 082900

```

var ch, ch2;

JSC$token_value = null;

while ((ch = stream.readByte ()) != -1)
{
    if (rjs_Error) return false;           //@@ avoid infinite loop

    if (ch == '\n')
    {
        JSC$linenum++;
        continue;
    }

    if (JSC$lexer_is_white_space (ch))
        continue;

    JSC$token_linenum = JSC$linenum;

    if (ch == '/' && JSC$lexer_peek_char (stream) == '*')
    {
        /* Multi line comment. */
        stream.readByte ();
        while ((ch = stream.readByte ()) != -1
            && (ch != '*' || JSC$lexer_peek_char (stream) != '/'))
            if (ch == '\n')
                JSC$linenum++;

        /* Consume the peeked '/' character. */
        stream.readByte ();
    }
    else if ((ch == '/' && JSC$lexer_peek_char (stream) == '/')
        || (ch == '#' && JSC$lexer_peek_char (stream) == '!'))
    {
        /* Single line comment. */
        while ((ch = stream.readByte ()) != -1 && ch != '\n')
            ;
        if (ch == '\n')
            JSC$linenum++;
    }
    else if (ch == '"' || ch == '\')
    {
        /* String constant. */
        JSC$token_value = JSC$lexer_read_string (stream, "string", ch);
        return JSC$tSTRING;
    }

    /* Literals. */
    else if (ch == '=' && JSC$lexer_peek_char (stream) == '=')
    {
        stream.readByte ();
        if (JSC$lexer_peek_char (stream) == '=')
        {
            stream.readByte ();
            return JSC$tSEQUAL;
        }
        return JSC$tEQUAL;
    }
}

```

```

    }
    else if (ch == '=' && JSC$lexer_peek_char (stream) == '=')
    {
        stream.readByte ();
        if (JSC$lexer_peek_char (stream) == '=')
        {
            stream.readByte ();
            return JSC$tSNEQUAL;
        }
        return JSC$tNEQUAL;
    }
    else if (ch == '<' && JSC$lexer_peek_char (stream) == '=')
    {
        stream.readByte ();
        return JSC$tLE;
    }
    else if (ch == '>' && JSC$lexer_peek_char (stream) == '=')
    {
        stream.readByte ();
        return JSC$tGE;
    }
    else if (ch == '&' && JSC$lexer_peek_char (stream) == '&')
    {
        stream.readByte ();
        return JSC$tAND;
    }
    else if (ch == '|' && JSC$lexer_peek_char (stream) == '|')
    {
        stream.readByte ();
        return JSC$tOR;
    }
    else if (ch == '+' && JSC$lexer_peek_char (stream) == '+')
    {
        stream.readByte ();
        return JSC$tPLUSPLUS;
    }
    else if (ch == '-' && JSC$lexer_peek_char (stream) == '-')
    {
        stream.readByte ();
        return JSC$tMINUSMINUS;
    }
    else if (ch == '*' && JSC$lexer_peek_char (stream) == '=')
    {
        stream.readByte ();
        return JSC$tMULA;
    }
    else if (ch == '/' && JSC$lexer_peek_char (stream) == '=')
    {
        stream.readByte ();
        return JSC$tDIVA;
    }
    else if (ch == '%' && JSC$lexer_peek_char (stream) == '=')
    {
        stream.readByte ();
        return JSC$tMODA;
    }
    else if (ch == '+' && JSC$lexer_peek_char (stream) == '=')

```

```

    {
        stream.readByte ();
        return JSC$tADDA;
    }
else if (ch == '-' && JSC$lexer_peek_char (stream) == '=')
    {
        stream.readByte ();
        return JSC$tSUBA;
    }
else if (ch == '&' && JSC$lexer_peek_char (stream) == '=')
    {
        stream.readByte ();
        return JSC$tAND;
    }
else if (ch == '^' && JSC$lexer_peek_char (stream) == '=')
    {
        stream.readByte ();
        return JSC$tXOR;
    }
else if (ch == '|' && JSC$lexer_peek_char (stream) == '=')
    {
        stream.readByte ();
        return JSC$tOR;
    }
else if (ch == '<' && JSC$lexer_peek_char (stream) == '<')
    {
        stream.readByte ();
        if (JSC$lexer_peek_char (stream) == '=')
            {
                stream.readByte ();
                return JSC$tLSIA;
            }
        else
            return JSC$tLSHIFT;
    }
else if (ch == '>' && JSC$lexer_peek_char (stream) == '>')
    {
        stream.readByte ();
        ch2 = JSC$lexer_peek_char (stream);
        if (ch2 == '=')
            {
                stream.readByte ();
                return JSC$tRSIA;
            }
        else if (ch2 == '>')
            {
                stream.readByte ();
                if (JSC$lexer_peek_char (stream) == '=')
                    {
                        stream.readByte ();
                        return JSC$tRRSA;
                    }
                else
                    return JSC$tRRSHIFT;
            }
        else
            return JSC$tRSHIFT;
    }

```

00650273.082900

```
    }

/* Identifiers and keywords. */
else if (JSC$lexer_is_identifier_letter (ch))
{
    /* An identifier. */
    //@@ var id = String.fromCharCode (ch);
    var id = "" + ch;

    while ((ch = stream.readByte ()) != -1
        && (JSC$lexer_is_identifier_letter (ch)
            || JSC$lexer_is_decimal_digit (ch)))
        id += ch;      //@@ id.append (File.toString (ch));

    stream.ungetByte (ch);

    /* Keywords. */
    if (id == "break")
        return JSC$tBREAK;
    else if (id == "continue")
        return JSC$tCONTINUE;
    else if (id == "delete")
        return JSC$tDELETE;
    else if (id == "else")
        return JSC$tELSE;
    else if (id == "for")
        return JSC$tFOR;
    else if (id == "function")
        return JSC$tFUNCTION;
    else if (id == "if")
        return JSC$tIF;
    else if (id == "in")
        return JSC$tIN;
    else if (id == "new")
        return JSC$tNEW;
    else if (id == "return")
        return JSC$tRETURN;
    else if (id == "this")
        return JSC$tTHIS;
    else if (id == "typeof")
        return JSC$tTYPEOF;
    else if (id == "var")
        return JSC$tVAR;
    else if (id == "void")
        return JSC$tVOID;
    else if (id == "while")
        return JSC$tWHILE;
    else if (id == "with")
        return JSC$tWITH;

    /*
     * Future reserved keywords (some of these is already in use
     * in this implementation).
     */
    else if (id == "case")
        return JSC$tCASE;
    else if (id == "catch")
```

00650273.082900

```
        return JSC$tCATCH;
    else if (id == "class")
        return JSC$tCLASS;
    else if (id == "const")
        return JSC$tCONST;
    else if (id == "debugger")
        return JSC$tDEBUGGER;
    else if (id == "default")
        return JSC$tDEFAULT;
    else if (id == "do")
        return JSC$tDO;
    else if (id == "enum")
        return JSC$tENUM;
    else if (id == "export")
        return JSC$tEXPORT;
    else if (id == "extends")
        return JSC$tEXTENDS;
    else if (id == "finally")
        return JSC$tFINALLY;
    else if (id == "import")
        return JSC$tIMPORT;
    else if (id == "super")
        return JSC$tSUPER;
    else if (id == "switch")
        return JSC$tSWITCH;
    else if (id == "throw")
        return JSC$tTHROW;
    else if (id == "try")
        return JSC$tTRY;

    /* Null and boolean literals. */
    else if (id == "null")
        return JSC$tNULL;
    else if (id == "true")
        return JSC$tTRUE;
    else if (id == "false")
        return JSC$tFALSE;
    else
    {
        /* It really is an identifier. */
        JSC$token_value = id;
        return JSC$tIDENTIFIER;
    }
}

/* Character constants. */
else if (ch == '#' && JSC$lexer_peek_char (stream) == '\\')
{
    /* Skip the starting '\\' and read more. */
    stream.readByte ();

    ch = stream.readByte ();
    if (ch == '\\')
    {
        JSC$token_value
            = JSC$lexer_read_backslash_escape (stream, 0, "character");
    }
}
```

```

        if (stream.readByte () != '\\')
            error (JSC$filename + ":" + JSC$linenum.toString ()
                + ": malformed character constant");
    }
    else if (JSC$lexer_peek_char (stream) == '\\')
    {
        stream.readByte ();
        JSC$token_value = ch;
    }
    else
        error (JSC$filename + ":" + JSC$linenum.toString ()
            + ": malformed character constant");

    return JSC$tINTEGER;
}

/* Octal and hex numbers. */
else if (ch == '0'
    && JSC$lexer_peek_char (stream) != '.'
    && JSC$lexer_peek_char (stream) != 'e'
    && JSC$lexer_peek_char (stream) != 'E')
{
    JSC$token_value = 0;
    ch = stream.readByte ();
    if (ch == 'x' || ch == 'X')
    {
        ch = stream.readByte ();
        while (JSC$lexer_is_hex_digit (ch))
        {
            JSC$token_value *= 16;
            JSC$token_value += JSC$lexer_hex_to_dec (ch);
            ch = stream.readByte ();
        }
        stream.ungetByte (ch);
    }
    else
    {
        while (JSC$lexer_is_octal_digit (ch))
        {
            JSC$token_value *= 8;
            JSC$token_value += ch - '0';
            ch = stream.readByte ();
        }
        stream.ungetByte (ch);
    }

    return JSC$tINTEGER;
}

/* Decimal numbers. */
else if (JSC$lexer_is_decimal_digit (ch)
    || (ch == '.'
        && JSC$lexer_is_decimal_digit (
            JSC$lexer_peek_char (stream))))
{
    var is_float = false;
    var buf = new String (File.byteToString (ch));

```

00650273.082900

```
var accept_dot = true;

if (ch == '.')
{
    /*
     * We started with '.' and we know that the next character
     * is a decimal digit (we peeked it).
     */
    is_float = true;

    ch = stream.readByte ();
    while (JSC$lexer_is_decimal_digit (ch))
    {
        buf += ch;    //@@ buf.append (File.toString (ch));
        ch = stream.readByte ();
    }
    accept_dot = false;
}
else
{
    /* We did start with a decimal digit. */
    ch = stream.readByte ();
    while (JSC$lexer_is_decimal_digit (ch))
    {
        buf += ch;    //@@ buf.append (File.toString (ch));
        ch = stream.readByte ();
    }
}

if ((accept_dot && ch == '.')
    || ch == 'e' || ch == 'E')
{
    is_float = true;

    if (ch == '.')
    {
        buf += ch;    //@@ buf.append (File.toString (ch));

        ch = stream.readByte ();
        while (JSC$lexer_is_decimal_digit (ch))
        {
            buf += ch;    //@@ buf.append (File.toString (ch));
            ch = stream.readByte ();
        }
    }

    if (ch == 'e' || ch == 'E')
    {
        buf += ch;    //@@ buf.append (File.toString (ch));
        ch = stream.readByte ();
        if (ch == '+' || ch == '-')
        {
            buf += ch;    //@@ buf.append (File.toString (ch));
            ch = stream.readByte ();
        }
        if (!JSC$lexer_is_decimal_digit (ch))
            error (JSC$filename + ":" + JSC$linenum.toString ())
    }
}
```



00650273.082900

```
        + ": malformed exponent part in a decimal literal");

        while (JSC$lexer_is_decimal_digit (ch))
        {
            buf += ch;    //@ buf.append (File.toString (ch));
            ch = stream.readByte ();
        }
    }

    /* Finally, we put the last character back to the stream. */
    stream.ungetByte (ch);

    if (is_float)
    {
        JSC$token_value = parseFloat (buf);
        return JSC$tFLOAT;
    }

    JSC$token_value = parseInt (buf);
    return JSC$tINTEGER;
}

/* Just return the character as-is. */
else
    return ch;
}

/* EOF reached. */
return JSC$tEOF;
}

/*
 * Help functions.
 */

function JSC$lexer_peek_char (stream)
{
    var ch2 = stream.readByte ();
    stream.ungetByte (ch2);

    return ch2;
}

function JSC$lexer_is_identifier_letter (ch)
{
    return (('a' <= ch && ch <= 'z') || ('A' <= ch && ch <= 'Z')
        || ch == '$' || ch == '_');
}

function JSC$lexer_is_octal_digit (ch)
{
    return ('0' <= ch && ch <= '7');
}
```

```
function JSC$lexer_is_decimal_digit (ch)
{
  return '0' <= ch && ch <= '9';
}
```

```
function JSC$lexer_is_hex_digit (ch)
{
  return (('0' <= ch && ch <= '9')
    || ('a' <= ch && ch <= 'f')
    || ('A' <= ch && ch <= 'F'));
}
```

```
function JSC$lexer_is_white_space (ch)
{
  //@@ return (ch == ' ' || ch == '\t' || ch == '\v' || ch == '\r'

  return (ch == ' ' || ch == '\t' || ch == rjs_VTAB || ch == '\r'
    || ch == '\f' || ch == '\n');
}
```

```
function JSC$lexer_hex_to_dec (ch)
{
  return (('0' <= ch && ch <= '9')
    ? ch - '0'
    : (('a' <= ch && ch <= 'f')
      ? 10 + ch - 'a'
      : 10 + ch - 'A'));
}
```

```
function JSC$lexer_read_backslash_escape (stream, possible_start, name)
{
  var ch = stream.readByte ();

  if (ch == 'n')
    ch = '\n';
  else if (ch == 't')
    ch = '\t';
  else if (ch == 'v')
    ch = rjs_VTAB;
  else if (ch == 'b')
    ch = '\b';
  else if (ch == 'r')
    ch = '\r';
  else if (ch == 'f')
    ch = '\f';
  else if (ch == 'a')
    ch = '\a';
  else if (ch == '\\')
    ch = '\\';
  else if (ch == '?')
    ch = '?';

  //@@ ch = '\v';
}
```

```

else if (ch == '\\')
    ch = '\\';
else if (ch == '"')
    ch = '"';
else if (ch == 'x')
{
    /* HexEscapeSequence. */
    var c1, c2;

    c1 = stream.readByte ();
    c2 = stream.readByte ();

    if (c1 == -1 || c2 == -1)
        JSC$lexer_eof_in_constant (possible_start, name);

    if (!JSC$lexer_is_hex_digit (c1) || !JSC$lexer_is_hex_digit (c2))
        error (JSC$filename + ":" + JSC$linenum.toString ()
            + ": \\x used with no following hex digits");

    ch = (JSC$lexer_hex_to_dec (c1) << 4) + JSC$lexer_hex_to_dec (c2);
}
else if (ch == 'u')
{
    /* UnicodeEscapeSequence. */
    var c1, c2, c3, c4;

    c1 = stream.readByte ();
    c2 = stream.readByte ();
    c3 = stream.readByte ();
    c4 = stream.readByte ();

    if (c1 == -1 || c2 == -1 || c3 == -1 || c4 == -1)
        JSC$lexer_eof_in_constant (possible_start, name);

    if (!JSC$lexer_is_hex_digit (c1) || !JSC$lexer_is_hex_digit (c2)
        || !JSC$lexer_is_hex_digit (c3) || !JSC$lexer_is_hex_digit (c4))
        error (JSC$filename + ":" + JSC$linenum.toString ()
            + ": \\u used with no following hex digits");

    ch = ((JSC$lexer_hex_to_dec (c1) << 12)
        + (JSC$lexer_hex_to_dec (c2) << 8)
        + (JSC$lexer_hex_to_dec (c3) << 4)
        + JSC$lexer_hex_to_dec (c4));
}
else if (JSC$lexer_is_octal_digit (ch))
{
    var result = ch - '0';
    var i = 1;

    if (ch == '0')
        /* Allow three octal digits after '0'. */
        i = 0;

    ch = stream.readByte ();
    while (i < 3 && JSC$lexer_is_octal_digit (ch))
    {
        result *= 8;
    }
}

```

006280" E205960

00660273 1082900

```
        result += ch - '0';
        ch = stream.readByte ();
        i++;
    }
    stream.ungetByte (ch);
    ch = result;
}
else
{
    if (ch == -1)
        error (JSC$filename + ":" + JSC$linenum.toString ()
            + ": unterminated " + name);

    JSC$warning (JSC$filename + ":" + JSC$linenum.toString ()
        + ": warning: unknown escape sequence \"\\\"
        + File.byteToString (ch) + "\"");
}

return ch;
}

function JSC$lexer_read_string (stream, name, ender)
{
    var str = new String ("");
    var done = false, ch;
    var possible_start_ln = JSC$linenum;
    var warned_line_terminator = false;

    while (!done)
    {
        if (rjs_Error) return false;           //@@ avoid infinite loop

        ch = stream.readByte ();
        if (ch == '\n')
        {
            if (JSC$warn_strict_ecma && !warned_line_terminator)
            {
                JSC$warning (JSC$filename + ":" + JSC$linenum.toString ()
                    + ": warning: ECMAScript don't allow line terminators
                    + name + " constants");
                warned_line_terminator = true;
            }
            JSC$linenum++;
        }

        if (ch == -1)
            JSC$lexer_eof_in_constant (possible_start_ln, name);

        else if (ch == ender)
            done = true;
        else
        {
            if (ch == '\\')
            {
                if (JSC$lexer_peek_char (stream) == '\n')

```

00650273 082900

```
        {
            /*
             * Backslash followed by a newline character.  Ignore
             * them both.
             */
            stream.readByte ();
            JSC$linenum++;
            continue;
        }
        ch = JSC$lexer_read_backslash_escape (stream, possible_start_ln,
                                              name);
    }
    str += ch;    //@@ str.append (ch);
}

return str;
}

function JSC$lexer_read_regexp_constant (stream)
{
    /* Regexp literal. */
    var source = JSC$lexer_read_regexp_source (stream);

    /* Check the possible flags. */
    var flags = new String ("");
    while ((ch = JSC$lexer_peek_char (stream)) == 'g' || ch == 'i')
    {
        stream.readByte ();
        flags += ch;    //@@ flags.append (File.toString (ch));
    }

    /* Try to compile it. */
    var msg = false;
    var result;

    //@@
    result = new RegExp (source, flags);

    /*** @@
    try
    {
        result = new RegExp (source, flags);
    }
    catch (msg)
    {
        var start = msg.lastIndexOf (":");
        msg = (JSC$filename + ":" + JSC$token_linenum.toString ()
              + ": malformed regular expression constant:"
              + msg.substr (start + 1));
    }
    ***/

    if (msg)
        error (msg);
}
```

006280" E2205960

```
/* Success. */
return result;
}

function JSC$lexer_read_regexp_source (stream)
{
    var str = new String ("");
    var done = false, ch;
    var possible_start_ln = JSC$linenum;
    var warned_line_terminator = false;
    var name = "regular expression";

    while (!done)
    {
        if (rjs_Error) return false;           //@@ avoid infinite loop

        ch = stream.readByte ();

        if (ch == '\n')
        {
            if (JSC$warn_strict_ecma && !warned_line_terminator)
            {
                JSC$warning (JSC$filename + ":" + JSC$linenum.toString ()
                    + ": warning: ECMAScript don't allow line "
                    + "terminators in " + name + " constants");
                warned_line_terminator = true;
            }
            JSC$linenum++;
        }

        if (ch == -1)
            JSC$lexer_eof_in_constant (possible_start_ln, name);

        else if (ch == '/')
            done = true;
        else
        {
            if (ch == '\\')
            {
                ch = stream.readByte ();
                if (ch == '\n')
                {
                    /*
                     * Backslash followed by a newline character.  Ignore
                     * them both.
                     */
                    JSC$linenum++;
                    continue;
                }
            }
            if (ch == -1)
                JSC$lexer_eof_in_constant (possible_start_ln, name);

            /* Handle the backslash escapes. */
            if (ch == 'f')
                ch = '\f';
        }
    }
}
```

00650273.082900

```

    else if (ch == 'n')
        ch = '\n';
    else if (ch == 'r')
        ch = '\r';
    else if (ch == 't')
        ch = '\t';
    else if (ch == 'v')
        ch = rjs_VTAB;          //@@ Bug with '==' from original codes?
ch == '\v';
    else if (ch == 'c')
    {
        /* SourceCharacter. */
        ch = stream.readByte ();
        if (ch == -1)
            JSC$lexer_eof_in_constant (possible_start_ln, name);

        if (ch == '\n' && JSC$warn_strict_ecma)
            JSC$warning (JSC$filename + ":" + JSC$linenum.toString ()
                + ": warning: ECMAScript don't allow line
termiantor after \\c in regular expression constants");

        /*
         * Append the source-character escape start. The ch
         * will be appended later.
         */
        str += "\\c";          //@@ str.append ("\\c");
    }
    else if (ch == 'u' || ch == 'x' || ch == '0')
    {
        /* These can be handled with the read_backslash_escape(). */
        stream.ungetByte (ch);
        ch = JSC$lexer_read_backslash_escape (stream);
    }
    else
    {
        /*
         * Nothing special. Leave it to the result as-is.
         * The regular expression package will handle it.
         */
        stream.ungetByte (ch);
        ch = '\\';
    }
    }
    str += ch;          //@@ str.append (File.toString (ch));
}

return str;
}

function JSC$lexer_eof_in_constant (possible_start, name)
{
    var msg = (JSC$filename + ":" + JSC$linenum.toString ()
        + ": unterminated " + name + " constant");

    if (possible_start > 0)

```

```

{
  //@@ msg += (System.lineBreakSequence
  msg += ("\n"
    + JSC$filename + ":" + possible_start.toString ()
    + ": possible real start of unterminated " + name + " constant");
}
error (msg);
}

```

006280"E2205360



```
/*  
Local variables:  
mode: c  
End:  
*/
```

00650273 082900

00650273.082900

```
/*
 * Parser.
 * Copyright (c) 1998 New Generation Software (NGS) Oy
 *
 * Author: Markku Rossi <mtr@ngs.fi>
 */

/*
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public
 * License along with this library; if not, write to the Free
 * Software Foundation, Inc., 59 Temple Place - Suite 330, Boston,
 * MA 02111-1307, USA
 */

/*
 * The GNU Library General Public License may also be downloaded at
 * http://www.gnu.org/copyleft/gpl.html.
 */

/*****
 *
 * This software was modified by Yahoo! Inc. under the terms
 * of the GNU Library General Public License (LGPL). For all
 * legal, copyright, and technical issues relating to how
 * this software can be used under GNU LGPL, please write to:
 *
 * GNU Compliance, Legal Dept., Yahoo! Inc.,
 * 3420 Central Expressway, Santa Clara, California U.S.A.
 *****/

/*
 * $Source: /usr/local/cvsroot/ngs/js/jsc/parser.js,v $
 * $Id: parser.js,v 1.26 1998/10/26 15:25:21 mtr Exp $
 */

/*
 * Global functions.
 */

function JSC$parser_reset ()
{
    JSC$function = null;
    JSC$global_stmts = null;
    JSC$nested_function_declarations = null;
}
```

```

function JSC$parser_parse (stream)
{
    JSC$linenum = 1;
    JSC$filename = stream.name;
    JSC$functions = new Array ();
    JSC$global_stmts = new Array ();
    JSC$nested_function_declarations = new Array ();
    JSC$anonymous_function_count = 0;
    JSC$parser_peek_token_valid = false;
    JSC$num_tokens = 0;
    JSC$num_arguments_identifiers = 0;
    JSC$num_missing_semicolons = 0;

    if (JSC$verbose)
        JSC$message ("jsc: parsing");

    while (JSC$parser_peek_token (stream) != JSC$EOF)
        if (!JSC$parser_parse_source_element (stream))
        {
            JSC$parser_syntax_error ();
            return false;                //@@ avoid infinite loop
        }

    if (JSC$verbose)
    {
        var msg = ("jsc: input stream had " + (JSC$linenum - 1).toString ()
            + " lines, " + JSC$num_tokens.toString () + " tokens");

        if (JSC$num_missing_semicolons > 0)
            msg += (" " + JSC$num_missing_semicolons.toString ()
                + " missing semicolons");

        JSC$message (msg);
    }
}

/*
 * General help functions.
 */

function JSC$parser_syntax_error ()
{
    error (JSC$filename + ":" + JSC$linenum.toString () + ": syntax error");
}

/* All warnings are reported through this function. */
function JSC$warning (line)
{
    rjs_warn(line);    //@@ System.stderr.writeln (line);
}

/* All messages are reported through this function. */
function JSC$message (line)

```

00650273 082900

```
{
    rjs_info(line);    //@@ System.stderr.writeln (line);
}

function JSC$parser_get_token (stream)
{
    JSC$num_tokens++;

    var token;
    if (JSC$parser_peek_token_valid)
    {
        JSC$parser_peek_token_valid = false;
        JSC$parser_token_value = JSC$parser_peek_token_value;
        JSC$parser_token_linenum = JSC$parser_peek_token_linenum;
        token = JSC$parser_peek_token_token;
    }
    else
    {
        token = JSC$lexer (stream);
        JSC$parser_token_value = JSC$token_value;
        JSC$parser_token_linenum = JSC$token_linenum;
    }

    if (token == JSC$tIDENTIFIER && JSC$parser_token_value == "arguments")
        JSC$num_arguments_identifiers++;

    return token;
}

function JSC$parser_peek_token (stream)
{
    if (JSC$parser_peek_token_valid)
        return JSC$parser_peek_token_token;
    else
    {
        JSC$parser_peek_token_token = JSC$lexer (stream);
        JSC$parser_peek_token_value = JSC$token_value;
        JSC$parser_peek_token_linenum = JSC$token_linenum;
        JSC$parser_peek_token_valid = true;
        return JSC$parser_peek_token_token;
    }
}

function JSC$parser_get_semicolon_ascii (stream)
{
    var token = JSC$parser_peek_token (stream);

    if (token == ';')
    {
        rjs_Tokens.push(";");    //@@

        /* Everything ok.  It was there. */
        return JSC$parser_get_token (stream);
    }
}
```

```

/* No semicolon. Let's see if we can insert it there. */
if (token == '}')
    || JSC$parser_token_linenum < JSC$parser_peek_token_linenum
    || token == JSC$tEOF)
{
    rjs_Tokens.push(";");    //@@

    /* Ok, do the automatic semicolon insertion. */
    if (JSC$warn_missing_semicolon)
        JSC$warning (JSC$filename + ":" + JSC$parser_token_linenum.toString ()
            + ": warning: missing semicolon");
    JSC$num_missing_semicolons++;
    return ' ';
}

/* Sorry, no can do. */
JSC$parser_syntax_error ();
}

function JSC$parser_expr_is_left_hand_side (expr)
{
    return (expr.etype == JSC$EXPR_CALL
        || expr.etype == JSC$EXPR_OBJECT_PROPERTY
        || expr.etype == JSC$EXPR_OBJECT_ARRAY
        || expr.etype == JSC$EXPR_NEW
        || expr.etype == JSC$EXPR_THIS
        || expr.etype == JSC$EXPR_IDENTIFIER
        || expr.etype == JSC$EXPR_FLOAT
        || expr.etype == JSC$EXPR_INTEGER
        || expr.etype == JSC$EXPR_STRING
        || expr.etype == JSC$EXPR_REGEXP
        || expr.etype == JSC$EXPR_ARRAY_INITIALIZER
        || expr.etype == JSC$EXPR_NULL
        || expr.etype == JSC$EXPR_TRUE
        || expr.etype == JSC$EXPR_FALSE);
}

function JSC$parser_parse_source_element (stream)
{
    rjs_Tokens.reset();    //@@

    if (JSC$parser_parse_function_declaration (stream))
    {
        rjs_Stmts.push( rjs_Tokens.str() );    //@@ save one statement
        return true;
    }

    rjs_Tokens.reset();    //@@

    var stmt = JSC$parser_parse_stmt (stream);

    if (!stmt)
        return false;
}

```

```

if (stmt.stype == JSC$STMT_VARIABLE)
/*
 * This is a variable declaration at the global level. These
 * are actually global variables.
 */
    stmt.global_level = true;

rjs_xDomain();           //@@
rjs_xLocation();         //@@
rjs_xCookie();           //@@
rjs_Stmts.push( rjs_Tokens.str() ); //@@ save one statement

JSC$global_stmts.push (stmt);

return true;
}

function JSC$parser_parse_function_declaration (stream)
{
    var id, args, block;

    if (JSC$parser_peek_token (stream) != JSC$tFUNCTION)
        return false;

    rjs_Tokens.push("function "); //@@

    /* Record how many `arguments' identifiers have been seen so far. */
    var num_arguments_identifiers = JSC$num_arguments_identifiers;

    JSC$parser_get_token (stream);
    if (JSC$parser_get_token (stream) != JSC$tIDENTIFIER)
        JSC$parser_syntax_error ();

    id = JSC$parser_token_value;
    var ln = JSC$parser_token_linenum;
    var id_given = id;

    rjs_Tokens.push(id_given); //@@

    if (JSC$nested_function_declarations.length > 0)
    {
        /* This is a nested function declaration. */
        id = ".F:" + (JSC$anonymous_function_count++).toString ();
    }
    JSC$nested_function_declarations.push (id);

    if (JSC$parser_get_token (stream) != '(')
        JSC$parser_syntax_error ();

    rjs_Tokens.push("("); //@@

    /* Formal parameter list opt. */
    args = new Array ();
    while (JSC$parser_peek_token (stream) != ')')
    {
        if (rjs_Error) return false; //@@ avoid infinite loop

```

```

    if (JSC$parser_get_token (stream) != JSC$tIDENTIFIER)
        JSC$parser_syntax_error ();
    args.push (JSC$parser_token_value);

    rjs_Tokens.push(JSC$parser_token_value);    //@@

    var token = JSC$parser_peek_token (stream);
    if (token == ',')
    {
        rjs_Tokens.push(",");    //@@

        JSC$parser_get_token (stream);
        if (JSC$parser_peek_token (stream) != JSC$tIDENTIFIER)
            JSC$parser_syntax_error ();
    }
    else if (token != ')')
        JSC$parser_syntax_error ();
}

if (JSC$parser_get_token (stream) != ')')
    JSC$parser_syntax_error ();

rjs_Tokens.push(") ");    //@@

JSC$parser_peek_token (stream);
var lbrace_ln = JSC$parser_peek_token_lineno;

block = JSC$parser_parse_block (stream);
if (typeof block == "boolean")
    JSC$parser_syntax_error ();

/* Did the function use the `arguments' identifier? */
var use_arguments = false;
if (JSC$num_arguments_identifiers > num_arguments_identifiers)
{
    use_arguments = true;
    if (JSC$warn_deprecated)
        JSC$warning (JSC$filename + ":" + ln.toString ()
            + ": warning: the `arguments' property of Function "
            + "instance is deprecated");
}

JSC$functions.push (new JSC$function_declaration (ln, lbrace_ln, id,
                                                    id_given, args,
                                                    block, use_arguments));

JSC$nested_function_declarations.pop ();

return true;
}

function JSC$parser_parse_block (stream)
{
    var block;

```

```

if (JSC$parser_peek_token (stream) != '{')
    return false;

//@@ original NGS bug ?? JSC$parser_get_token (stream) != '{';
JSC$parser_get_token (stream);

rjs_Tokens.push("{}"); //@@

var ln = JSC$parser_peek_token_linenum;

/* Do we have a statement list? */
if (JSC$parser_peek_token (stream) != '}')
    /* Yes we have. */
    block = JSC$parser_parse_stmt_list (stream);
else
    /* Do we don't */
    block = new Array ();

if (JSC$parser_get_token (stream) != '}')
    JSC$parser_syntax_error ();

rjs_Tokens.push("{}"); //@@

block.linenum = ln;

return block;
}

function JSC$parser_parse_stmt_list (stream)
{
    var list, done, item;

    list = new Array ();
    done = false;

    while (!done)
    {
        if (rjs_Error) return false; //@@ avoid infinite loop

        item = JSC$parser_parse_stmt (stream);
        if (typeof item == "boolean")
        {
            /* Can't parse more statements. We'r done. */
            done = true;
        }
        else
            list.push (item);
    }

    return list;
}

function JSC$parser_parse_stmt (stream)
{
    var item, token;

```



```

if (typeof (item = JSC$parser_parse_block (stream)) != "boolean")
    return new JSC$stmt_block (item.linenum, item);
else if (JSC$parser_parse_function_declaration (stream))
{
    /**
     * XXX The function declaration as statement might be incomplete. */

    if (JSC$nested_function_declarations.length == 0)
        /* Function declaration at top-level statements. */
        return new JSC$stmt_empty (JSC$parser_token_linenum);

    /* Function declaration inside another function. */

    var container_id = JSC$nested_function_declarations.pop ();
    JSC$nested_function_declarations.push (container_id);

    var f = JSC$functions[JSC$functions.length - 1];
    var function_id = f.name;
    var given_id = f.name_given;

    return new JSC$stmt_function_declaration (JSC$parser_token_linenum,
                                              container_id, function_id,
                                              given_id);
}
else if (typeof (item = JSC$parser_parse_variable_stmt (stream))
        != "boolean")
    return item;
else if (typeof (item = JSC$parser_parse_if_stmt (stream))
        != "boolean")
    return item;
else if (typeof (item = JSC$parser_parse_iteration_stmt (stream))
        != "boolean")
    return item;
else if (typeof (item = JSC$parser_parse_expr (stream))
        != "boolean")
{
    if (item.etype == JSC$EXPR_IDENTIFIER)
    {
        /* Possible 'Labeled Statement'. */
        token = JSC$parser_peek_token (stream);
        if (token == ':' && item.linenum == JSC$parser_peek_token_linenum)
        {
            /* Yes it is. */
            JSC$parser_get_token (stream);

            rjs_Tokens.push(": ");          /**

            var stmt = JSC$parser_parse_stmt (stream);
            if (!stmt)
                JSC$parser_syntax_error;

            return new JSC$stmt_labeled_stmt (item.linenum, item.value,
                                              stmt);
        }
        /* FALLTHROUGH */
    }
}

```

```

JSC$parser_get_semicolon_asci (stream);

return new JSC$stmt_expr (item);
}
else
{
    token = JSC$parser_peek_token (stream);
    if (token == ';')
    {
        rjs_Tokens.push(";");    //@@

        JSC$parser_get_token (stream);
        return new JSC$stmt_empty (JSC$parser_token_linenum);
    }
    else if (token == JSC$tCONTINUE)
    {
        rjs_Tokens.push("continue ");    //@@

        JSC$parser_get_token (stream);

        /* Check the possible label. */
        var label = null;
        token = JSC$parser_peek_token (stream);
        if (token == JSC$tIDENTIFIER
            && JSC$parser_token_linenum == JSC$parser_peek_token_linenum)
        {
            JSC$parser_get_token (stream);
            label = JSC$parser_token_value;

            rjs_Tokens.push(label);    //@@
        }

        item = new JSC$stmt_continue (JSC$parser_token_linenum, label);

        JSC$parser_get_semicolon_asci (stream);

        return item;
    }
    else if (token == JSC$tBREAK)
    {
        JSC$parser_get_token (stream);

        rjs_Tokens.push("break ");    //@@

        /* Check the possible label. */
        var label = null;
        token = JSC$parser_peek_token (stream);
        if (token == JSC$tIDENTIFIER
            && JSC$parser_token_linenum == JSC$parser_peek_token_linenum)
        {
            JSC$parser_get_token (stream);
            label = JSC$parser_token_value;

            rjs_Tokens.push(label);    //@@
        }
    }
}

```

00650273 082900

```

        item = new JSC$stmt_break (JSC$parser_token_linenum, label);

        JSC$parser_get_semicolon_asci (stream);

        return item;
    }
    else if (token == JSC$tRETURN)
    {
        JSC$parser_get_token (stream);
        var linenum = JSC$parser_token_linenum;

        rjs_Tokens.push("return ");    //@@

        if (JSC$parser_peek_token (stream) == ';')
        {
            /* Consume the semicolon. */
            JSC$parser_get_token (stream);
            item = null;

            rjs_Tokens.push(";");    //@@
        }
        else
        {
            if (JSC$parser_peek_token_linenum > linenum)
            {
                /*
                 * A line terminator between tRETURN and the next
                 * token that is not a semicolon.  ASCI here.
                 */
                if (JSC$warn_missing_semicolon)
                    JSC$warning (JSC$filename + ":" + linenum.toString ()
                                + ": warning: missing semicolon");

                JSC$num_missing_semicolons++;
                item = null;
            }
            else
            {
                item = JSC$parser_parse_expr (stream);
                if (typeof item == "boolean")
                    JSC$parser_syntax_error ();

                JSC$parser_get_semicolon_asci (stream);
            }
        }

        return new JSC$stmt_return (linenum, item);
    }
    else if (token == JSC$tSWITCH)    //@@
    {
        JSC$parser_get_token (stream);
        return JSC$parser_parse_switch (stream);
    }
    else if (token == JSC$tWITH)
    {
        rjs_Tokens.push("with ");    //@@

```

```

JSC$parser_get_token (stream);
var linenum = JSC$parser_token_linenum;

if (JSC$parser_get_token (stream) != '(')
    JSC$parser_syntax_error ();

rjs_Tokens.push("(");    //@@

var expr = JSC$parser_parse_expr (stream);
if (typeof expr == "boolean")
    JSC$parser_syntax_error ();

if (JSC$parser_get_token (stream) != ')')
    JSC$parser_syntax_error ();

rjs_Tokens.push(")");    //@@

var stmt = JSC$parser_parse_stmt (stream);
if (typeof stmt == "boolean")
    JSC$parser_syntax_error ();

return new JSC$stmt_with (linenum, expr, stmt);
}
else if (token == JSC$tTRY)    //@@
{
    JSC$parser_get_token (stream);
    return JSC$parser_parse_try (stream);
}
else if (token == JSC$tTHROW)    //@@
{
    JSC$parser_get_token (stream);
    var linenum = JSC$parser_token_linenum;

    /*
     * Get the next token's linenum. We need it for strict_ecma
     * warning.
     */
    JSC$parser_peek_token (stream);
    var peek_linenum = JSC$parser_peek_token_linenum;

    /* The expression to throw. */
    var expr = JSC$parser_parse_expr (stream);
    if (typeof expr == "boolean")
        JSC$parser_syntax_error ();

    if (JSC$warn_strict_ecma && peek_linenum > linenum)
        JSC$warning (JSC$filename + ":" + JSC$linenum.toString ()
            + ": warning: ECMAScript don't allow line terminators"
            + " between `throw' and expression");

    JSC$parser_get_semicolon_ascii (stream);

    return new JSC$stmt_throw (linenum, expr);
}
else
    /* Can't parse more. We'r done. */
    return false;

```

```

    }
}

```

```

function JSC$parser_parse_switch (stream)
{
    var linenum = JSC$parser_token_linenum;

    if (JSC$parser_get_token (stream) != '(')
        JSC$parser_syntax_error ();

    var expr = JSC$parser_parse_expr (stream);
    if (!expr)
        JSC$parser_syntax_error ();

    if (JSC$parser_get_token (stream) != ')')
        JSC$parser_syntax_error ();

    if (JSC$parser_get_token (stream) != '{')
        JSC$parser_syntax_error ();

    /* Parse case clauses. */
    var clauses = new Array ();
    while (true)
    {
        if (rjs_Error) return false;           //@@ avoid infinite loop

        var token = JSC$parser_get_token (stream);

        if (token == '}')
            break;
        else if (token == JSC$tCASE || token == JSC$tDEFAULT)
        {
            var stmts = new Array ();
            stmts.expr = null;

            if (token == JSC$tCASE)
            {
                stmts.expr = JSC$parser_parse_expr (stream);
                if (!stmts.expr)
                    JSC$parser_syntax_error ();
            }
            if (JSC$parser_get_token (stream) != ':')
                JSC$parser_syntax_error ();

            stmts.linenum = JSC$parser_token_linenum;

            /* Read the statement list. */
            while (true)
            {
                if (rjs_Error) return false;           //@@ avoid infinite loop

                token = JSC$parser_peek_token (stream);
                if (token == '}' || token == JSC$tCASE || token == JSC$tDEFAULT)
                    /* Done with this branch. */
                    break;
            }
        }
    }
}

```

```

        var stmt = JSC$parser_parse_stmt (stream);
        if (!stmt)
            JSC$parser_syntax_error ();

        stmts.push (stmt);
    }

    stmts.last_linenum = JSC$parser_token_linenum;

    /* One clause parsed. */
    clauses.push (stmts);
}
else
    JSC$parser_syntax_error ();
}

return new JSC$stmt_switch (linenum, JSC$parser_token_linenum, expr,
                             clauses);
}

function JSC$parser_parse_try (stream)
{
    var linenum = JSC$parser_token_linenum;

    var block = JSC$parser_parse_stmt (stream);
    if (!block)
        JSC$parser_syntax_error ();

    var try_block_last_linenum = JSC$parser_token_linenum;

    /* Now we must see `catch' or `finally'. */
    var token = JSC$parser_peek_token (stream);
    if (token != JSC$tCATCH && token != JSC$tFINALLY)
        JSC$parser_syntax_error ();

    var catch_list = false;
    if (token == JSC$tCATCH)
    {
        /* Parse catch list. */

        catch_list = new Array ();
        catch_list.linenum = JSC$parser_peek_token_linenum;

        while (token == JSC$tCATCH)
        {
            if (rjs_Error) return false;           //@@ avoid infinite loop

            JSC$parser_get_token (stream);
            var c = new Object ();
            c.linenum = JSC$parser_token_linenum;

            if (JSC$parser_get_token (stream) != '(')
                JSC$parser_syntax_error ();

            if (JSC$parser_get_token (stream) != JSC$tIDENTIFIER)
                JSC$parser_syntax_error ();

```

```

    c.id = JSC$parser_token_value;

    c.guard = false;
    if (JSC$parser_peek_token (stream) == JSC$tIF)
    {
        JSC$parser_get_token (stream);
        c.guard = JSC$parser_parse_expr (stream);
        if (!c.guard)
            JSC$parser_syntax_error ();
    }

    if (JSC$parser_get_token (stream) != ')')
        JSC$parser_syntax_error ();

    c.stmt = JSC$parser_parse_stmt (stream);
    if (!c.stmt)
        JSC$parser_syntax_error ();

    catch_list.push (c);

    token = JSC$parser_peek_token (stream);
}

catch_list.last_linenum = JSC$parser_token_linenum;
}

var fin = false;
if (token == JSC$tFINALLY)
{
    /* Parse the finally. */
    JSC$parser_get_token (stream);

    fin = JSC$parser_parse_stmt (stream);
    if (!fin)
        JSC$parser_syntax_error ();
}

return new JSC$stmt_try (linenum, try_block_last_linenum,
                        JSC$parser_token_linenum, block, catch_list,
                        fin);
}

function JSC$parser_parse_variable_stmt (stream)
{
    var list, id, expr, token;

    if (JSC$parser_peek_token (stream) != JSC$tVAR)
        return false;

    JSC$parser_get_token (stream);
    var ln = JSC$parser_token_linenum;

    rjs_Tokens.push("var "); //@@

    list = new Array ();

```

```

while (true)
{
    if (rjs_Error) return false;          //@@ avoid infinite loop

    token = JSC$parser_peek_token (stream);
    if (token == JSC$tIDENTIFIER)
    {
        JSC$parser_get_token ();
        id = JSC$parser_token_value;

        rjs_Tokens.push(id);    //@@

        if (JSC$parser_peek_token (stream) == '=')
        {
            rjs_Tokens.push("=");    //@@

            JSC$parser_get_token (stream);
            expr = JSC$parser_parse_assignment_expr (stream);
            if (typeof expr == "boolean")
                JSC$parser_syntax_error ();
        }
        else
            expr = null;

        list.push (new JSC$var_declaration (id, expr));
        // @@ rjs_debug("JSC$parser_parse_variable_stmt: var " + id + " = " +
        expr.value);

        /* Check if we have more input. */
        if (JSC$parser_peek_token (stream) == ',')
        {
            /* Yes we have. */
            JSC$parser_get_token (stream);

            rjs_Tokens.push(",");    //@@

            /* The next token must be tIDENTIFIER. */
            if (JSC$parser_peek_token (stream) != JSC$tIDENTIFIER)
                JSC$parser_syntax_error ();
        }
        else
        {
            /* No, we don't. */
            JSC$parser_get_semicolon_ascii (stream);
            break;
        }
    }
    else
    {
        /* We'r done. */
        JSC$parser_get_semicolon_ascii (stream);
        break;
    }
}

/* There must be at least one variable declaration. */

```



```

    if (list.length == 0)
        JSC$parser_syntax_error ();

    return new JSC$stmt_variable (ln, list);
}

function JSC$parser_parse_if_stmt (stream)
{
    var expr, stmt, stmt2;

    if (JSC$parser_peek_token (stream) != JSC$tIF)
        return false;

    rjs_Tokens.push(" if ");    //@@

    JSC$parser_get_token (stream);
    var ln = JSC$parser_token_linenum;

    if (JSC$parser_get_token (stream) != '(')
        JSC$parser_syntax_error ();

    rjs_Tokens.push("(");    //@@

    expr = JSC$parser_parse_expr (stream);
    if (typeof expr == "boolean")
        JSC$parser_syntax_error ();

    if (JSC$parser_get_token (stream) != ')')
        JSC$parser_syntax_error ();

    rjs_Tokens.push(" ");    //@@

    stmt = JSC$parser_parse_stmt (stream);
    if (typeof stmt == "boolean")
        JSC$parser_syntax_error ();

    if (JSC$parser_peek_token (stream) == JSC$tELSE)
    {
        rjs_Tokens.push(" else ");    //@@

        JSC$parser_get_token (stream);
        stmt2 = JSC$parser_parse_stmt (stream);
        if (typeof stmt2 == "boolean")
            JSC$parser_syntax_error ();
    }
    else
        stmt2 = null;

    return new JSC$stmt_if (ln, expr, stmt, stmt2);
}

function JSC$parser_parse_iteration_stmt (stream)
{
    var token, expr1, expr2, expr3, stmt;

```

```

token = JSC$parser_peek_token (stream);
if (token == JSC$tDO)
{
    rjs_Tokens.push(" do ");    //@@

    /* do Statement while (Expression); */
    JSC$parser_get_token (stream);
    var ln = JSC$parser_token_linenum;

    stmt = JSC$parser_parse_stmt (stream);
    if (typeof stmt == "boolean")
        JSC$parser_syntax_error ();

    if (JSC$parser_get_token (stream) != JSC$tWHILE)
        JSC$parser_syntax_error ();

    rjs_Tokens.push(" while ");    //@@

    if (JSC$parser_get_token (stream) != '(')
        JSC$parser_syntax_error ();

    rjs_Tokens.push("(");    //@@

    expr1 = JSC$parser_parse_expr (stream);
    if (typeof expr1 == "boolean")
        JSC$parser_syntax_error ();

    if (JSC$parser_get_token (stream) != ')')
        JSC$parser_syntax_error ();

    rjs_Tokens.push(")");    //@@

    JSC$parser_get_semicolon_asci (stream);

    return new JSC$stmt_do_while (ln, expr1, stmt);
}
else if (token == JSC$tWHILE)
{
    rjs_Tokens.push(" while ");    //@@

    /* while (Expression) Statement */
    JSC$parser_get_token (stream);
    var ln = JSC$parser_token_linenum;

    if (JSC$parser_get_token (stream) != '(')
        JSC$parser_syntax_error ();

    rjs_Tokens.push(" ( ");    //@@

    expr1 = JSC$parser_parse_expr (stream);
    if (typeof expr1 == "boolean")
        JSC$parser_syntax_error ();

    if (JSC$parser_get_token (stream) != ')')
        JSC$parser_syntax_error ();

    rjs_Tokens.push(" ) ");    //@@
}

```

```

stmt = JSC$parser_parse_stmt (stream);
if (typeof stmt == "boolean")
    JSC$parser_syntax_error ();

return new JSC$stmt_while (ln, expr1, stmt);
}
else if (token == JSC$tFOR)
{
    rjs_Tokens.push(" for ");    //@@

    JSC$parser_get_token (stream);
    var ln = JSC$parser_token_linenum;

    if (JSC$parser_get_token (stream) != '(')
        JSC$parser_syntax_error ();

    rjs_Tokens.push("(");    //@@

    /* Init */

    var vars = null;

    token = JSC$parser_peek_token (stream);
    if (token == JSC$tVAR)
    {
        JSC$parser_get_token (stream);

        rjs_Tokens.push("var ");    //@@

        vars = new Array ();

        while (true)
        {
            if (rjs_Error) return false;    //@@ avoid infinite loop

            /* The identifier. */
            token = JSC$parser_peek_token (stream);
            if (token != JSC$tIDENTIFIER)
                break;

            JSC$parser_get_token (stream);
            var id = JSC$parser_token_value;

            rjs_Tokens.push(id);    //@@

            /* Possible initializer. */
            var expr = null;
            if (JSC$parser_peek_token (stream) == '=')
            {
                JSC$parser_get_token (stream);

                rjs_Tokens.push("=");    //@@

                expr = JSC$parser_parse_assignment_expr (stream);
                if (!expr)
                    JSC$parser_syntax_error ();
            }
        }
    }
}

```

```

    }

    vars.push (new JSC$var_declaration (id, expr));

    /* Check if we have more input. */
    if (JSC$parser_peek_token (stream) == ',')
    {
        /* Yes we have. */
        JSC$parser_get_token (stream);

        rjs_Tokens.push(",");    //@@

        /* The next token must be tIDENTIFIER. */
        if (JSC$parser_peek_token (stream) != JSC$tIDENTIFIER)
            JSC$parser_syntax_error ();
    }
    else
        /* No more input. */
        break;
}

/* Must have at least one variable declaration. */
if (vars.length == 0)
    JSC$parser_syntax_error ();
}
else if (token != ';')
{
    expr1 = JSC$parser_parse_expr (stream);
    if (typeof expr1 == "boolean")
        JSC$parser_syntax_error ();
}
else
    expr1 = null;

token = JSC$parser_get_token (stream);
var for_in = false;

if (token == ';')
{
    rjs_Tokens.push(";");    //@@

    /* Normal for-statement. */

    /* Check */
    if (JSC$parser_peek_token (stream) != ';')
    {
        expr2 = JSC$parser_parse_expr (stream);
        if (typeof expr2 == "boolean")
            JSC$parser_syntax_error ();
    }
    else
        expr2 = null;

    if (JSC$parser_get_token (stream) != ';')
        JSC$parser_syntax_error ();
}

```

```

    rjs_Tokens.push(";");    //@@

    /* Increment */
    if (JSC$parser_peek_token (stream) != ')')
    {
        expr3 = JSC$parser_parse_expr (stream);
        if (typeof expr3 == "boolean")
            JSC$parser_syntax_error ();
    }
    else
        expr3 = null;
}
else if (token == JSC$tIN)
{
    /* The `for (VAR in EXPR)'-statement. */

    rjs_Tokens.push(" in ");    //@@

    for_in = true;

    if (expr1)
    {
        /* The first expression must be an identifier. */
        if (expr1.etype != JSC$EXPR_IDENTIFIER)
            JSC$parser_syntax_error ();
    }
    else
    {
        /* We must have only one variable declaration. */
        if (vars.length != 1)
            JSC$parser_syntax_error ();
    }

    /* The second expressions. */
    expr2 = JSC$parser_parse_expr (stream);
    if (typeof expr2 == "boolean")
        JSC$parser_syntax_error ();
}
else
    JSC$parser_syntax_error ();

if (JSC$parser_get_token (stream) != ')')
    JSC$parser_syntax_error ();

rjs_Tokens.push(" ");    //@@

/* Stmt. */
stmt = JSC$parser_parse_stmt (stream);
if (typeof stmt == "boolean")
    JSC$parser_syntax_error ();

if (for_in)
    return new JSC$stmt_for_in (ln, vars, expr1, expr2, stmt);

return new JSC$stmt_for (ln, vars, expr1, expr2, expr3, stmt);
}
return false;

```

[illegible]

}

```
var str = "";

if (rjs_isEndOfLHS("location") || rjs_isEndOfLHS("location.href"))
{
    str = rjs_xUrlBegin( rjs_t2s(token) + "rmi_xlateURL(" );
    rjs_XUrl_nesting.push(0); // for tracking '('
}
else if (rjs_isEndOfLHS(".action"))
{
    str = rjs_xActionBegin( rjs_t2s(token) + "rmi_xlateURL(" );
    rjs_XAction_nesting.push(0); // for tracking '('
}
else if (rjs_isEndOfLHS(".innerHTML"))
{
    str = rjs_xInnerHtmlBegin( rjs_t2s(token) + "rmi_xlate(" );
    rjs_XInnerHtml_nesting.push(0); // for tracking '('
}
else if (rjs_isEndOfLHS("document.cookie"))
{
    // @@ rule: document.cookie = cookieStr
    str = rjs_xCookieBegin( "rmi_setCookie(\"\\\", \" )");
    rjs_XCookie_nesting.push(0); // for tracking '('
}
else
    str = rjs_t2s(token);

rjs_Tokens.push(str);
rjs_AssignmentState = "rhs";
rjs_popDomain();
rjs_popLocation();
rjs_popCookie();

//@@ <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

JSC$parser_get_token (stream);
var ln = JSC$parser_token_linenum;

expr2 = JSC$parser_parse_assignment_expr (stream);
if (typeof expr2 == "boolean")
    JSC$parser_syntax_error ();

expr = new JSC$expr_assignment (ln, token, expr, expr2);

$optimize_constant_folding && expr.constant_folding)
n expr.constant_folding ();

le In translation state and no more unmatched '('
_Xurl_on && rjs_retTop(rjs_XUrl_nesting) == 0 )

js_Tokens.push( rjs_xUrlEnd( ")" ) );
js_XUrl_nesting.pop(); // no need to track '(' any
```

```

// @@rule In translation state and no more unmatched '('
if (rjs_XCookie_on && rjs_retTop(rjs_XCookie_nesting) == 0 )
{
    rjs_Tokens.push( rjs_xCookieEnd( ")" ) );
    rjs_XCookie_nesting.pop(); // no need to track '(' any
more
}

// @@rule In translation state and no more unmatched '('
if (rjs_XAction_on && rjs_retTop(rjs_XAction_nesting) == 0 )
{
    rjs_Tokens.push( rjs_xActionEnd( ")" ) );
    rjs_XAction_nesting.pop(); // no need to track '(' any
more
}

// @@rule In translation state and no more unmatched '('
if (rjs_XInnerHTML_on && rjs_retTop(rjs_XInnerHTML_nesting) == 0 )
{
    rjs_Tokens.push( rjs_xInnerHTMLEnd( ")" ) );
    rjs_XInnerHTML_nesting.pop(); // no need to track '('
any more
}

return expr;
}

function JSC$parser_parse_conditional_expr (stream)
{
    var expr, expr2, expr3, token;

    if (typeof (expr = JSC$parser_parse_logical_or_expr (stream))
        == "boolean")
        return false;

    token = JSC$parser_peek_token (stream);
    if (token == '?')
    {
        rjs_Tokens.push("?"); //@@

        JSC$parser_get_token (stream);
        var ln = JSC$parser_token_linenum;

        expr2 = JSC$parser_parse_assignment_expr (stream);
        if (typeof expr2 == "boolean")
            JSC$parser_syntax_error ();

        if (JSC$parser_get_token (stream) != ':')
            JSC$parser_syntax_error ();

        rjs_Tokens.push(":"); //@@

        expr3 = JSC$parser_parse_assignment_expr (stream);
        if (typeof expr3 == "boolean")
            JSC$parser_syntax_error ();
    }

```



00650273.082900

```
    expr = new JSC$expr_quest_colon (ln, expr, expr2, expr3);
  }

  return expr;
}

function JSC$parser_parse_logical_or_expr (stream)
{
  var expr, expr2;

  if (typeof (expr = JSC$parser_parse_logical_and_expr (stream))
      == "boolean")
    return false;

  while (JSC$parser_peek_token (stream) == JSC$tOR)
  {
    if (rjs_Error) return false;          //@@ avoid infinite loop

    rjs_Tokens.push("||");                //@@

    JSC$parser_get_token (stream);
    var ln = JSC$parser_token_linenum;

    expr2 = JSC$parser_parse_logical_and_expr (stream);
    if (typeof expr2 == "boolean")
      JSC$parser_syntax_error ();

    expr = new JSC$expr_logical_or (ln, expr, expr2);
  }

  return expr;
}

function JSC$parser_parse_logical_and_expr (stream)
{
  var expr, expr2;

  if (typeof (expr = JSC$parser_parse_bitwise_or_expr (stream))
      == "boolean")
    return false;

  while (JSC$parser_peek_token (stream) == JSC$tAND)
  {
    if (rjs_Error) return false;          //@@ avoid infinite loop

    rjs_Tokens.push("&&");                //@@

    JSC$parser_get_token (stream);
    var ln = JSC$parser_token_linenum;

    expr2 = JSC$parser_parse_bitwise_or_expr (stream);
    if (typeof expr2 == "boolean")
      JSC$parser_syntax_error ();
  }
}
```

```

        expr = new JSC$expr_logical_and (ln, expr, expr2);
    }

    return expr;
}

function JSC$parser_parse_bitwise_or_expr (stream)
{
    var expr, expr2;

    if (typeof (expr = JSC$parser_parse_bitwise_xor_expr (stream))
        == "boolean")
        return false;

    while (JSC$parser_peek_token (stream) == '|')
    {
        if (rjs_Error) return false;           //@@ avoid infinite loop

        rjs_Tokens.push("|");           //@@

        JSC$parser_get_token (stream);
        var ln = JSC$parser_token_linenum;

        expr2 = JSC$parser_parse_bitwise_xor_expr (stream);
        if (typeof expr2 == "boolean")
            JSC$parser_syntax_error ();

        expr = new JSC$expr_bitwise_or (ln, expr, expr2);
    }

    return expr;
}

function JSC$parser_parse_bitwise_xor_expr (stream)
{
    var expr, expr2;

    if (typeof (expr = JSC$parser_parse_bitwise_and_expr (stream))
        == "boolean")
        return false;

    while (JSC$parser_peek_token (stream) == '^')
    {
        if (rjs_Error) return false;           //@@ avoid infinite loop
        rjs_Tokens.push("^");           //@@

        JSC$parser_get_token (stream);
        var ln = JSC$parser_token_linenum;

        expr2 = JSC$parser_parse_bitwise_and_expr (stream);
        if (typeof expr2 == "boolean")
            JSC$parser_syntax_error ();

        expr = new JSC$expr_bitwise_xor (ln, expr, expr2);
    }
}

```

```

    return expr;
}

function JSC$parser_parse_bitwise_and_expr (stream)
{
    var expr, expr2;

    if (typeof (expr = JSC$parser_parse_equality_expr (stream))
        == "boolean")
        return false;

    while (JSC$parser_peek_token (stream) == '&')
    {
        if (rjs_Error) return false;          //@@ avoid infinite loop
        rjs_Tokens.push("&");                  //@@

        JSC$parser_get_token (stream);
        var ln = JSC$parser_token_linenum;

        expr2 = JSC$parser_parse_equality_expr (stream);
        if (typeof expr2 == "boolean")
            JSC$parser_syntax_error ();

        expr = new JSC$expr_bitwise_and (ln, expr, expr2);
    }

    return expr;
}

function JSC$parser_parse_equality_expr (stream)
{
    var expr, expr2, token;

    if (typeof (expr = JSC$parser_parse_relational_expr (stream))
        == "boolean")
        return false;

    token = JSC$parser_peek_token (stream);
    while (token == JSC$tEQUAL || token == JSC$tNEQUAL
        || token == JSC$tSEQUAL || token == JSC$tSNEQUAL)
    {
        if (rjs_Error) return false;          //@@ avoid infinite loop
        rjs_Tokens.push(rjs_t2s(token) );    //@@

        JSC$parser_get_token (stream);
        var ln = JSC$parser_token_linenum;

        expr2 = JSC$parser_parse_relational_expr (stream);
        if (typeof expr2 == "boolean")
            JSC$parser_syntax_error ();

        expr = new JSC$expr_equality (ln, token, expr, expr2);
        token = JSC$parser_peek_token (stream);
    }
}

```

```

    return expr;
}

```

```

function JSC$parser_parse_relational_expr (stream)
{
    var expr, expr2, token;

    if (typeof (expr = JSC$parser_parse_shift_expr (stream))
        == "boolean")
        return false;

    token = JSC$parser_peek_token (stream);
    while (token == '<' || token == '>' || token == JSC$tLE
        || token == JSC$tGE)
    {
        if (rjs_Error) return false;          //@@ avoid infinite loop
        rjs_Tokens.push(rjs_t2s(token) );      //@@

        JSC$parser_get_token (stream);
        var ln = JSC$parser_token_linenum;

        expr2 = JSC$parser_parse_shift_expr (stream);
        if (typeof expr2 == "boolean")
            JSC$parser_syntax_error ();

        expr = new JSC$expr_relational (ln, token, expr, expr2);
        token = JSC$parser_peek_token (stream);
    }

    return expr;
}

```

```

function JSC$parser_parse_shift_expr (stream)
{
    var expr, expr2, token;

    if (typeof (expr = JSC$parser_parse_additive_expr (stream))
        == "boolean")
        return false;

    token = JSC$parser_peek_token (stream);
    while (token == JSC$tLSHIFT || token == JSC$tRSHIFT || token == JSC$tRRSHIFT)
    {
        if (rjs_Error) return false;          //@@ avoid infinite loop
        rjs_Tokens.push(rjs_t2s(token) );      //@@

        JSC$parser_get_token (stream);
        var ln = JSC$parser_token_linenum;

        expr2 = JSC$parser_parse_additive_expr (stream);

        if (typeof expr2 == "boolean")
            JSC$parser_syntax_error ();
    }
}

```

```

    expr = new JSC$expr_shift (ln, token, expr, expr2);
    token = JSC$parser_peek_token (stream);
}

return expr;
}

function JSC$parser_parse_additive_expr (stream)
{
    var expr, expr2, token;

    if (typeof (expr = JSC$parser_parse_multiplicative_expr (stream))
        == "boolean")
        return false;

    token = JSC$parser_peek_token (stream);
    while (token == '+' || token == '-')
    {
        if (rjs_Error) return false;           //@@ avoid infinite loop
        rjs_Tokens.push(token);                //@@

        JSC$parser_get_token (stream);
        var ln = JSC$parser_token_linenum;

        expr2 = JSC$parser_parse_multiplicative_expr (stream);
        if (typeof expr2 == "boolean")
            JSC$parser_syntax_error ();

        expr = new JSC$expr_additive (ln, token, expr, expr2);
        token = JSC$parser_peek_token (stream);
    }

    return expr;
}

function JSC$parser_parse_multiplicative_expr (stream)
{
    var expr, expr2, token;

    if (typeof (expr = JSC$parser_parse_unary_expr (stream)) == "boolean")
        return false;

    token = JSC$parser_peek_token (stream);
    while (token == '*' || token == '/' || token == '%')
    {
        if (rjs_Error) return false;           //@@ avoid infinite loop
        rjs_Tokens.push(token);                //@@

        JSC$parser_get_token (stream);
        var ln = JSC$parser_token_linenum;

        expr2 = JSC$parser_parse_unary_expr (stream);
        if (typeof expr2 == "boolean")
            JSC$parser_syntax_error ();
    }

```

```

        expr = new JSC$expr_multiplicative (ln, token, expr, expr2);
        token = JSC$parser_peek_token (stream);
    }

    return expr;
}

function JSC$parser_parse_unary_expr (stream)
{
    var expr, token;

    token = JSC$parser_peek_token (stream);
    if (token == JSC$tDELETE
        || token == JSC$tVOID
        || token == JSC$tTYPEOF
        || token == JSC$tPLUSPLUS
        || token == JSC$tMINUSMINUS
        || token == '+'
        || token == '-'
        || token == '~'
        || token == '!')
    {
        rjs_Tokens.push(rjs_t2s(token));    //@@

        JSC$parser_get_token (stream);
        var ln = JSC$parser_token_linenum;

        expr = JSC$parser_parse_unary_expr (stream);
        if (typeof expr == "boolean")
            JSC$parser_syntax_error ();

        return new JSC$expr_unary (ln, token, expr);
    }

    return JSC$parser_parse_postfix_expr (stream);
}

function JSC$parser_parse_postfix_expr (stream)
{
    var expr, token;

    if (typeof (expr = JSC$parser_parse_left_hand_side_expr (stream))
        == "boolean")
        return false;

    token = JSC$parser_peek_token (stream);
    if (token == JSC$tPLUSPLUS || token == JSC$tMINUSMINUS)
    {
        if (JSC$parser_peek_token_linenum > JSC$parser_token_linenum)
        {
            if (JSC$warn_missing_semicolon)
                JSC$warning (JSC$filename + ":"
                    + JSC$parser_token_linenum.toString ()
                    + ": warning: automatic semicolon insertion cuts the
expression before ++ or --");
        }
    }

```

```

    }
    else
    {
        rjs_Tokens.push(rjs_t2s(token));    //@@

        JSC$parser_get_token (stream);
        var ln = JSC$parser_token_linenum;

        return new JSC$expr_postfix (ln, token, expr);
    }
}

return expr;
}

function JSC$parser_parse_left_hand_side_expr (stream)
{
    var expr, args, token, expr2;

    if (typeof (expr = JSC$parser_parse_member_expr (stream))
        == "boolean")
        return false;

    /* Parse the possible first pair of arguments. */
    if (JSC$parser_peek_token (stream) == '(')
    {
        var ln = JSC$parser_peek_token_linenum;

        args = JSC$parser_parse_arguments (stream);
        if (typeof args == "boolean")
            JSC$parser_syntax_error ();

        expr = new JSC$expr_call (ln, expr, args);
    }
    else
        return expr;

    /* Parse to possibly following arguments and selectors. */
    while ((token = JSC$parser_peek_token (stream)) == '('
        || token == '[' || token == '.')
    {
        if (rjs_Error) return false;    //@@ avoid infinite loop

        var ln = JSC$parser_peek_token_linenum;

        if (token == '(')
        {
            args = JSC$parser_parse_arguments (stream);
            expr = new JSC$expr_call (ln, expr, args);
        }
        else if (token == '[')
        {
            rjs_Tokens.push(" " + token);    //@@

            JSC$parser_get_token (stream);

```

```

    expr2 = JSC$parser_parse_expr (stream);
    if (typeof expr2 == "boolean")
        JSC$parser_syntax_error ();

    if (JSC$parser_get_token (stream) != ']')
        JSC$parser_syntax_error ();

    rjs_Tokens.push("]"); //@@

    expr = new JSC$expr_object_array (ln, expr, expr2);
}
else
{
    rjs_Tokens.push(" " + token); // token == '.' //@@

    JSC$parser_get_token (stream);
    if (JSC$parser_get_token (stream) != JSC$tIDENTIFIER)
        JSC$parser_syntax_error ();

    rjs_Tokens.push(" " + JSC$parser_token_value); //@@

    expr = new JSC$expr_object_property (ln, expr,
                                          JSC$parser_token_value);
}
}

return expr;
}

function JSC$parser_parse_member_expr (stream)
{
    var expr, args, token, expr2;

    if (typeof (expr = JSC$parser_parse_primary_expr (stream))
        == "boolean")
    {
        token = JSC$parser_peek_token (stream);

        if (token == JSC$tNEW)
        {
            rjs_Tokens.push("new"); //@@

            JSC$parser_get_token (stream);
            var ln = JSC$parser_token_linenum;

            expr = JSC$parser_parse_member_expr (stream);
            if (typeof expr == "boolean")
                JSC$parser_syntax_error ();

            if (JSC$parser_peek_token (stream) == '(')
            {
                args = JSC$parser_parse_arguments (stream);
                if (typeof args == "boolean")
                    JSC$parser_syntax_error ();
            }
        }
        else
    }

```



```

        return new JSC$expr_new (ln, expr, null);

    expr = new JSC$expr_new (ln, expr, args);
    }
    else
        return false;
    }

/* Ok, now we have valid starter. */
token = JSC$parser_peek_token (stream);
while (token == '[' || token == '.')
{
    if (rjs_Error) return false;           //@@ avoid infinite loop

    JSC$parser_get_token (stream);
    var ln = JSC$parser_token_linenum;

    if (token == '[')
    {
        rjs_Tokens.push("[");           //@@
        rjs_incTopForNesting();         //@@ see [
        rjs_BracketState = "in";        //@@
        rjs_saveFrames();               //@@ rule

        expr2 = JSC$parser_parse_expr (stream);
        if (typeof expr2 == "boolean")
            JSC$parser_syntax_error ();

        if (JSC$parser_get_token (stream) != ']')
            JSC$parser_syntax_error ();

        rjs_Tokens.push("]");           //@@
        rjs_xFrames();                 //@@ rule
        rjs_BracketState = "out";       //@@
        rjs_decTopForNesting();         //@@ see ]

        expr = new JSC$expr_object_array (ln, expr, expr2);
    }
    else
    {
        rjs_Tokens.push(".");           // token == '.' //@@

        if (JSC$parser_get_token (stream) != JSC$tIDENTIFIER)
            JSC$parser_syntax_error ();

        rjs_Tokens.push(JSC$parser_token_value); //@@
        rjs_xLayers(JSC$parser_token_value); //@@ rule
        rjs_saveDomain();               //@@ rule
        // rjs_saveLocation();           //@@ rule

        expr = new JSC$expr_object_property (ln, expr,
                                              JSC$parser_token_value);
    }

    token = JSC$parser_peek_token (stream);

    rjs_saveLocation();                 //@@ rule
}

```

09650273.082900

```

        rjs_saveCookie();                                //@@ rule
    }

    rjs_saveStandaloneLocation();    //@@ rule

    return expr;
}

function JSC$parser_parse_primary_expr (stream)
{
    rjs_debug("JSC$parser_parse_primary_exp");    //@@

    var token, val;

    token = JSC$parser_peek_token (stream);
    var ln = JSC$parser_peek_token_linenum;

    if (token == JSC$tTHIS)
    {
        rjs_Tokens.push("this");    //@@
        val = new JSC$expr_this (ln);
    }
    else if (token == JSC$tIDENTIFIER)
    {
        val = new JSC$expr_identifier (ln, JSC$parser_peek_token_value);

        rjs_Tokens.push(JSC$parser_peek_token_value);    //@@
        rjs_debug("JSC$tIDENTIFIER: " + JSC$parser_peek_token_value + ", " +
rjs_AssignmentState );    //@@

        if (JSC$parser_peek_token_value == "document")        //@@ rule
            rjs_LayerState = "doc";

        if (rjs_BracketState != "in")        //@@ If not in []
            rjs_saveIndexFor("id");        //@@ save current identifier index
    }
    else if (token == JSC$tFLOAT)        //@@
    {
        rjs_Tokens.push(JSC$parser_peek_token_value);        //@@
        val = new JSC$expr_float (ln, JSC$parser_peek_token_value);
    }
    else if (token == JSC$tINTEGER)        //@@
    {
        rjs_Tokens.push(JSC$parser_peek_token_value);        //@@
        val = new JSC$expr_integer (ln, JSC$parser_peek_token_value);
    }
    else if (token == JSC$tSTRING)
    {
        rjs_Tokens.push("\\"" + JSC$parser_peek_token_value + "\"");    //@@
        val = new JSC$expr_string (ln, JSC$parser_peek_token_value);
    }
    else if (token == '/')        //@@
    {
        /*

```



```

var expr = JSC$parser_parse_assignment_expr (stream);
if (!expr)
    JSC$parser_syntax_error ();

items[pos] = expr;

/* Got one expression. It must be followed by ',' or ']' */
token = JSC$parser_peek_token (stream);
if (token != ',' && token != ']')
    JSC$parser_syntax_error ();

}

if (token == ']') rjs_Tokens.push("");          //@@

val = new JSC$expr_array_initializer (ln, items);
}
else if (token == '{')
{
    /* Object literal. */
    /* TODO: SharpVarDefinition_{opt} */

    rjs_Tokens.push('{');                      //@@
    JSC$parser_get_token (stream);

    var items = new Array ();

    while ((token = JSC$parser_peek_token (stream)) != '}')
    {
        if (rjs_Error) return false;          //@@ avoid infinite loop

        var pair = new Object ();

        token = JSC$parser_get_token (stream);

        pair.linenum = JSC$linenum;
        pair.id_type = token;
        pair.id = JSC$parser_token_value;

        if (token != JSC$tIDENTIFIER && token != JSC$tSTRING
            && token != JSC$tINTEGER)
            JSC$parser_syntax_error ();

        if (token == JSC$tSTRING) rjs_Tokens.push "\"" + pair.id + "\"");
//@@
        else if (token == JSC$tIDENTIFIER) rjs_Tokens.push(pair.id);
//@@

        if (JSC$parser_get_token (stream) != ':')
            JSC$parser_syntax_error ();

        rjs_Tokens.push(':');                  //@@

        pair.expr = JSC$parser_parse_assignment_expr (stream);
        if (!pair.expr)
            JSC$parser_syntax_error ();

```

```

    items.push (pair);

    /*
     * Got one property, initializer pair. It must be followed
     * by ',' or '}'.
     */
    token = JSC$parser_peek_token (stream);
    if (token == ',')
    {
        rjs_Tokens.push(',');    //@@

        /* Ok, we have more items. */
        JSC$parser_get_token (stream);

        token = JSC$parser_peek_token (stream);
        if (token != JSC$tIDENTIFIER && token != JSC$tSTRING
            && token != JSC$tINTEGER)
            JSC$parser_syntax_error ();
    }
    else if (token != '}' && token)
        JSC$parser_syntax_error ();
}
if (token == '}') rjs_Tokens.push("{}");    //@@

val = new JSC$expr_object_initializer (ln, items);
}
else if (token == '(')
{
    rjs_Tokens.push("(");    //@@
    rjs_incTopForNesting();    //@@ see (

    JSC$parser_get_token (stream);

    val = JSC$parser_parse_expr (stream);
    if (typeof val == "boolean"
        || JSC$parser_peek_token (stream) != ')')
        JSC$parser_syntax_error ();

    rjs_Tokens.push("");    //@@
    rjs_decTopForNesting();    //@@ see )
}
else
    return false;

JSC$parser_get_token (stream);

//@@ rjs_debug("JSC$parser_parse_primary_expr: " + val['value'] );

return val;
}

function JSC$parser_parse_arguments (stream)
{
    var args, item;

```

```

if (JSC$parser_peek_token (stream) != '(')
    return false;

args = new Array ();

rjs_Tokens.push("(");    //@@
rjs_saveOpen();          //@@
rjs_saveWrite();          //@@
rjs_saveReplace();        //@@
rjs_incTopForNesting();   //@@ see (

JSC$parser_get_token (stream);
while (JSC$parser_peek_token (stream) != ')')
{
    if (rjs_Error) return false;    //@@ avoid infinite loop

    item = JSC$parser_parse_assignment_expr (stream);
    if (typeof item == "boolean")
        JSC$parser_syntax_error ();
    args.push (item);

    var token = JSC$parser_peek_token (stream);
    if (token == ',')
        JSC$parser_get_token (stream);
    else if (token != ')')
        JSC$parser_syntax_error ();

    if (token == ')')
    {
        rjs_xOpen();            //@@ rule
        rjs_xWrite();           //@@ rule
        rjs_xReplace();         //@@ rule

        rjs_decTopForNesting();  //@@ see )
    }

    rjs_Tokens.push("" + token);  //@@
}

if (token != ')')
{
    rjs_decTopForNesting();    // will insert )
    rjs_Tokens.push(")");     // take care of ()    //@@
}

JSC$parser_get_token (stream);

return args;
}

```

```
/*  
Local variables:  
mode: c  
End:  
*/
```

09650273.082900

```

/*
 * Grammar components.
 * Copyright (c) 1998 New Generation Software (NGS) Oy
 *
 * Author: Markku Rossi <mtr@ngs.fi>
 */

/*
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public
 * License along with this library; if not, write to the Free
 * Software Foundation, Inc., 59 Temple Place - Suite 330, Boston,
 * MA 02111-1307, USA
 */

/*
 * The GNU Library General Public License may also be downloaded at
 * http://www.gnu.org/copyleft/gpl.html.
 */

/*****
 *
 * This software was modified by Yahoo! Inc. under the terms
 * of the GNU Library General Public License(LGPL). For all
 * legal, copyright, and technical issues relating to how
 * this software can be used under GNU LGPL, please write to:
 *
 * GNU Compliance, Legal Dept., Yahoo! Inc.,
 * 3420 Central Expressway, Santa Clara, California U.S.A.
 *
 *****/

/* @@
 * Remove this.asm = *;
 * Remove function JSC$_asm () {...}
 */

/*
 * $Source: /usr/local/cvsroot/ngs/js/jsc/gram.js,v $
 * $Id: gram.js,v 1.22 1998/10/26 15:25:21 mtr Exp $
 */

/* General helpers. */

function JSC$gram_reset ()
{

```



```

JSC$label_count = 1;
JSC$cont_break = new JSC$ContBreak ();
}

function JSC$alloc_label (num_labels)
{
    JSC$label_count += num_labels;

    return JSC$label_count - num_labels;
}

function JSC$format_label (num)
{
    return ".L" + num.toString ();
}

function JSC$count_locals_from_stmt_list (list)
{
    var i;

    /* First, count how many variables we need at the toplevel. */
    var lcount = 0;
    for (i = 0; i < list.length; i++)
        lcount += list[i].count_locals (false);

    /* Second, count the maximum amount needed by the nested blocks. */
    var rmax = 0;
    for (i = 0; i < list.length; i++)
    {
        var rc = list[i].count_locals (true);
        if (rc > rmax)
            rmax = rc;
    }

    return lcount + rmax;
}

/*
 * The handling of the `continue' and `break' labels for looping
 * constructs. The variable `JSC$cont_break' holds an instance of
 * JSC$ContBreak class. The instance contains a valid chain of
 * looping constructs and the currently active with and try testing
 * levels. The actual `continue', `break', and `return' statements
 * investigate the chain and generate appropriate `with_pop' and
 * `try_pop' operands.
 *
 * If the instance variable `inswitch' is true, the continue statement
 * is inside a switch statement. In this case, the continue statement
 * must pop one item from the stack. That item is the value of the
 * case expression.
 */

function JSC$ContBreakFrame (loop_break, loop_continue, inswitch, label, next)
{

```

```

    this.loop_break = loop_break;
    this.loop_continue = loop_continue;
    this.inswitch = inswitch;
    this.label = label;
    this.next = next;

    this.with_nesting = 0;
    this.try_nesting = 0;
}

function JSC$ContBreak ()
{
    this.top = new JSC$ContBreakFrame (null, null, false, null);
}

new JSC$ContBreak ();

function JSC$ContBreak$push (loop_break, loop_continue, inswitch, label)
{
    this.top = new JSC$ContBreakFrame (loop_break, loop_continue, inswitch,
                                         label, this.top);
}
JSC$ContBreak.prototype.push = JSC$ContBreak$push;

function JSC$ContBreak$pop ()
{
    if (this.top == null)
        error ("jsc: internal error: continue-break stack underflow");

    this.top = this.top.next;
}
JSC$ContBreak.prototype.pop = JSC$ContBreak$pop;

/*
 * Count the currently active `try' nesting that should be removed on
 * `return' statement.
 */
function JSC$ContBreak$try_return_nesting ()
{
    var f;
    var count = 0;

    for (f = this.top; f; f = f.next)
        count += f.try_nesting;

    return count;
}
JSC$ContBreak.prototype.try_return_nesting = JSC$ContBreak$try_return_nesting;

/*
 * Count currently active `with' nesting that should be removed on
 * `continue' or `break' statement.
 */
function JSC$ContBreak$count_with_nesting (label)
{
    var f;

```

```

var count = 0;

for (f = this.top; f; f = f.next)
{
    count += f.with_nesting;
    if (label)
    {
        if (f.label == label)
            break;
    }
    else
        if (f.loop_continue)
            break;
}
return count;
}
JSC$ContBreak.prototype.count_with_nesting = JSC$ContBreak$count_with_nesting;

/*
 * Count the currently active `try' nesting that should be removed on
 * `continue' or `break' statement.
 */
function JSC$ContBreak$count_try_nesting (label)
{
    var f;
    var count = 0;

    for (f = this.top; f; f = f.next)
    {
        count += f.try_nesting;
        if (label)
        {
            if (f.label == label)
                break;
        }
        else
            if (f.loop_continue)
                break;
    }
    return count;
}
JSC$ContBreak.prototype.count_try_nesting = JSC$ContBreak$count_try_nesting;

function JSC$ContBreak$count_switch_nesting (label)
{
    var f;
    var count = 0;

    for (f = this.top; f; f = f.next)
    {
        if (f.inswitch)
            count++;
        if (label)
        {
            if (f.label == label)
                break;
        }
    }
}

```

```

        else
            if (f.loop_continue)
                break;
        }
        return count;
    }
    JSC$ContBreak.prototype.count_switch_nesting
        = JSC$ContBreak$count_switch_nesting;

function JSC$ContBreak$get_continue (label)
{
    var f;

    for (f = this.top; f; f = f.next)
        if (label)
        {
            if (f.label == label)
                return f.loop_continue;
        }
        else
            if (f.loop_continue)
                return f.loop_continue;

    return null;
}
JSC$ContBreak.prototype.get_continue = JSC$ContBreak$get_continue;

function JSC$ContBreak$get_break (label)
{
    var f;

    for (f = this.top; f; f = f.next)
        if (label)
        {
            if (f.label == label)
                return f.loop_break;
        }
        else
            if (f.loop_break)
                return f.loop_break;

    return null;
}
JSC$ContBreak.prototype.get_break = JSC$ContBreak$get_break;

function JSC$ContBreak$is_unique_label (label)
{
    var f;

    for (f = this.top; f; f = f.next)
        if (f.label == label)
            return false;

    return true;
}
JSC$ContBreak.prototype.is_unique_label = JSC$ContBreak$is_unique_label;

```

```
JSC$cont_break = null;
```

```
/* Function declaration. */
```

```
function JSC$function_declaration (ln, lbrace_ln, name, name_given, args,
                                   block, use_arguments_prop)
```

```
{
  this.linenum = ln;
  this.lbrace_linenum = lbrace_ln;
  this.name = name;
  this.name_given = name_given;
  this.args = args;
  this.block = block;
  this.use_arguments_prop = use_arguments_prop;
}
```

```
function JSC$zero_function ()
```

```
{
  return 0;
}
```

```
/*
 * Statements.
 */
```

```
/* Block. */
```

```
function JSC$stmt_block (ln, list)
```

```
{
  rjs_debug("JSC$stmt_block:");

  this.stype = JSC$STMT_BLOCK;
  this.linenum = ln;
  this.stmts = list;
  this.count_locals = JSC$stmt_block_count_locals;
}
```

```
function JSC$stmt_block_count_locals (recursive)
```

```
{
  if (!recursive)
    return 0;

  return JSC$count_locals_from_stmt_list (this.stmts);
}
```

```
/* Function declaration. */
```

```
function JSC$stmt_function_declaration (ln, container_id, function_id,
                                         given_id)
```

```
{
  rjs_debug("JSC$stmt_function_declaration:");

  this.stype = JSC$STMT_FUNCTION_DECLARATION;
  this.linenum = ln;
  this.container_id = container_id;
}
```

```

    this.function_id = function_id;
    this.given_id = given_id;
    this.count_locals = JSC$zero_function;
}

/* Empty */

function JSC$stmt_empty (ln)
{
    this.stype = JSC$STMT_EMPTY;
    this.linenum = ln;
    this.count_locals = JSC$zero_function;
}

/* Continue. */

function JSC$stmt_continue (ln, label)
{
    rjs_debug("JSC$stmt_continue:");

    this.stype = JSC$STMT_CONTINUE;
    this.linenum = ln;
    this.label = label;
    this.count_locals = JSC$zero_function;
}

/* Break. */

function JSC$stmt_break (ln, label)
{
    rjs_debug(" JSC$stmt_break:");

    this.stype = JSC$STMT_BREAK;
    this.linenum = ln;
    this.label = label;
    this.count_locals = JSC$zero_function;
}

/* Return. */

function JSC$stmt_return (ln, expr)
{
    rjs_debug("JSC$stmt_return:");

    this.stype = JSC$STMT_RETURN;
    this.linenum = ln;
    this.expr = expr;
    this.count_locals = JSC$zero_function;
}

/* Switch. */

function JSC$stmt_switch (ln, last_ln, expr, clauses)
{
    rjs_debug("JSC$stmt_switch:");

    this.stype = JSC$STMT_SWITCH;

```

005200 00273 002900

```
this.linenum = ln;
this.last_linenum = last_ln;
this.expr = expr;
this.clauses = clauses;
this.count_locals = JSC$stmt_switch_count_locals;
}

function JSC$stmt_switch_count_locals (recursive)
{
    var locals = 0;
    var i, j;

    if (recursive)
    {
        /* For the recursive cases, we need the maximum of our clause stmts. */
        for (i = 0; i < this.clauses.length; i++)
        {
            var c = this.clauses[i];
            for (j = 0; j < c.length; j++)
            {
                var l = c[j].count_locals (true);
                if (l > locals)
                    locals = l;
            }
        }
    }
    else
    {
        /*
         * The case clauses are not blocks. Therefore, we need the amount,
         * needed by the clauses at the top-level.
         */

        for (i = 0; i < this.clauses.length; i++)
        {
            var c = this.clauses[i];
            for (j = 0; j < c.length; j++)
                locals += c[j].count_locals (false);
        }
    }

    return locals;
}

/* With. */

function JSC$stmt_with (ln, expr, stmt)
{
    rjs_debug("JSC$stmt_with:");

    this.stype = JSC$STMT_WITH;
    this.linenum = ln;
    this.expr = expr;
    this.stmt = stmt;
    this.count_locals = JSC$stmt_with_count_locals;
}
```

```

function JSC$stmt_with_count_locals (recursive)
{
    if (!recursive)
    {
        if (this.stmt.stype == JSC$STMT_VARIABLE)
            return this.stmt.list.length;

        return 0;
    }
    else
        return this.stmt.count_locals (true);
}

/* Try. */

function JSC$stmt_try (ln, try_block_last_ln, try_last_ln, block, catch_list,
                        fin)
{
    rjs_debug("JSC$stmt_try:");

    this.stype = JSC$STMT_TRY;
    this.linenum = ln;
    this.try_block_last_linenum = try_block_last_ln;
    this.try_last_linenum = try_last_ln;
    this.block = block;
    this.catch_list = catch_list;
    this.fin = fin;
    this.count_locals = JSC$stmt_try_count_locals;
}

function JSC$stmt_try_count_locals (recursive)
{
    var count = 0;
    var c;

    if (recursive)
    {
        c = this.block.count_locals (true);
        if (c > count)
            count = c;

        if (this.catch_list)
        {
            var i;
            for (i = 0; i < this.catch_list.length; i++)
            {
                c = this.catch_list[i].stmt.count_locals (true);
                if (c > count)
                    count = c;
            }
        }
    }
    if (this.fin)
    {
        c = this.fin.count_locals (true);
        if (c > count)

```



```

        count = c;
    }
}
else
{
    if (this.block.stype == JSC$STMT_VARIABLE)
        count += this.block.list.length;

    if (this.catch_list)
    {
        /* One for the call variable. */
        count++;

        var i;
        for (i = 0; i < this.catch_list.length; i++)
            if (this.catch_list[i].stmt.stype == JSC$STMT_VARIABLE)
                count += this.catch_list[i].stmt.list.length;
    }

    if (this.fin)
        if (this.fin.stype == JSC$STMT_VARIABLE)
            count += this.fin.list.length;
}

return count;
}

/* Throw. */
function JSC$stmt_throw (ln, expr)
{
    rjs_debug("JSC$stmt_throw:");

    this.stype = JSC$STMT_THROW;
    this.linenum = ln;
    this.expr = expr;
    this.count_locals = JSC$zero_function;
}

/* Labeled statement. */
function JSC$stmt_labeled_stmt (ln, id, stmt)
{
    rjs_debug("JSC$stmt_labeled_stmt:");

    this.stype = JSC$STMT_LABELED_STMT;
    this.linenum = ln;
    this.id = id;
    this.stmt = stmt;
    this.count_locals = JSC$stmt_labeled_stmt_count_locals;
}

function JSC$stmt_labeled_stmt_count_locals (recursive)
{
    return this.stmt.count_locals (recursive);
}

```

```
/* Expression. */
```

```
function JSC$stmt_expr (expr)
{
  rjs_debug("JSC$stmt_expr:");

  this.stype = JSC$STMT_EXPR;
  this.linenum = expr.linenum;
  this.expr = expr;
  this.count_locals = JSC$zero_function;
}
```

```
/* If. */
```

```
function JSC$stmt_if (ln, expr, stmt1, stmt2)
{
  rjs_debug("JSC$stmt_if:");

  this.stype = JSC$STMT_IF;
  this.linenum = ln;
  this.expr = expr;
  this.stmt1 = stmt1;
  this.stmt2 = stmt2;
  this.count_locals = JSC$stmt_if_count_locals;
}
```

```
function JSC$stmt_if_count_locals (recursive)
{
  var lcount;

  if (!recursive)
  {
    lcount = 0;
    if (this.stmt1.stype == JSC$STMT_VARIABLE)
      lcount += this.stmt1.list.length;

    if (this.stmt2 != null && this.stmt2.stype == JSC$STMT_VARIABLE)
      lcount += this.stmt2.list.length;
  }
  else
  {
    lcount = this.stmt1.count_locals (true);

    if (this.stmt2)
    {
      var c = this.stmt2.count_locals (true);
      if (c > lcount)
        lcount = c;
    }
  }

  return lcount;
}
```

```
/* Do...While. */
```

```

function JSC$stmt_do_while (ln, expr, stmt)
{
  rjs_debug("JSC$stmt_do_while:");

  this.stype = JSC$STMT_DO_WHILE;
  this.linenum = ln;
  this.expr = expr;
  this.stmt = stmt;
  this.count_locals = JSC$stmt_do_while_count_locals;
}

function JSC$stmt_do_while_count_locals (recursive)
{
  if (!recursive)
  {
    if (this.stmt.stype == JSC$STMT_VARIABLE)
      return this.stmt.list.length;

    return 0;
  }
  else
    return this.stmt.count_locals (true);
}

/* While. */

function JSC$stmt_while (ln, expr, stmt)
{
  rjs_debug("JSC$stmt_while:");

  this.stype = JSC$STMT_WHILE;
  this.linenum = ln;
  this.expr = expr;
  this.stmt = stmt;
  this.count_locals = JSC$stmt_while_count_locals;
}

function JSC$stmt_while_count_locals (recursive)
{
  if (!recursive)
  {
    if (this.stmt.stype == JSC$STMT_VARIABLE)
      return this.stmt.list.length;

    return 0;
  }
  else
    return this.stmt.count_locals (true);
}

/* For. */

function JSC$stmt_for (ln, vars, e1, e2, e3, stmt)
{
  rjs_debug("JSC$stmt_for:");

```

```

    this.stype = JSC$STMT_FOR;
    this.linenum = ln;
    this.vars = vars;
    this.expr1 = e1;
    this.expr2 = e2;
    this.expr3 = e3;
    this.stmt = stmt;
    this.count_locals = JSC$stmt_for_count_locals;
}

function JSC$stmt_for_count_locals (recursive)
{
    var count = 0;

    if (recursive)
    {
        if (this.vars)
            count += this.vars.length;

        count += this.stmt.count_locals (true);
    }
    else
    {
        if (this.stmt.stype == JSC$STMT_VARIABLE)
            count += this.stmt.list.length;
    }

    return count;
}

/* For...in. */

function JSC$stmt_for_in (ln, vars, e1, e2, stmt)
{
    rjs_debug("JSC$stmt_for_in:");

    this.stype = JSC$STMT_FOR_IN;
    this.linenum = ln;
    this.vars = vars;
    this.expr1 = e1;
    this.expr2 = e2;
    this.stmt = stmt;
    this.count_locals = JSC$stmt_for_in_count_locals;
}

function JSC$stmt_for_in_count_locals (recursive)
{
    var count = 0;

    if (recursive)
    {
        if (this.vars)
            count++;

        count += this.stmt.count_locals (true);
    }
}

```

```

    }
    else
    {
        if (this.stmt.stype == JSC$STMT_VARIABLE)
            count += this.stmt.list.length;
    }

    return count;
}

/* Variable. */

function JSC$stmt_variable (ln, list)
{
    this.stype = JSC$STMT_VARIABLE;
    this.linenum = ln;
    this.global_level = false;
    this.list = list;
    this.count_locals = JSC$stmt_variable_count_locals;
}

function JSC$stmt_variable_count_locals (recursive)
{
    if (!recursive)
    {
        if (this.global_level)
            /* We define these as global variables. */
            return 0;

        return this.list.length;
    }

    return 0;
}

function JSC$var_declaration (id, expr)
{
    rjs_debug("JSC$var_declaration - " + id);

    this.id = id;
    this.expr = expr;
}

/*
 * Expressions.
 */

/* This. */

function JSC$expr_this (ln)
{
    rjs_debug("JSC$expr_this:");

    this.etype = JSC$EXPR_THIS;
}

```

```
    this.linenum = ln;
}

/* Identifier. */

function JSC$expr_identifier (ln, value)
{
    rjs_debug("JSC$expr_identifier:" + value);

    this.etype = JSC$EXPR_IDENTIFIER;
    this.linenum = ln;
    this.value = value;
}

/* Float. */

function JSC$expr_float (ln, value)
{
    rjs_debug("JSC$expr_float:");

    this.etype = JSC$EXPR_FLOAT;
    this.lang_type = JSC$JS_FLOAT;
    this.linenum = ln;
    this.value = value;
}

/* Integer. */

function JSC$expr_integer (ln, value)
{
    rjs_debug("JSC$expr_integer:");

    this.etype = JSC$EXPR_INTEGER;
    this.lang_type = JSC$JS_INTEGER;
    this.linenum = ln;
    this.value = value;
}

/* String. */

function JSC$expr_string (ln, value)
{
    rjs_debug("JSC$expr_string:" + value);

    this.etype = JSC$EXPR_STRING;
    this.lang_type = JSC$JS_STRING;
    this.linenum = ln;
    this.value = value;
}

/* Regexp. */

function JSC$expr_regexp (ln, value)
{
    rjs_debug("JSC$expr_regexp:");
```

```

    this.etype = JSC$EXPR_REGEX;
    this.lang_type = JSC$JS_BUILTIN;
    this.linenum = ln;
    this.value = value;
}

/* Array initializer. */

function JSC$expr_array_initializer (ln, items)
{
    rjs_debug("JSC$expr_array_initializer:");

    this.etype = JSC$EXPR_ARRAY_INITIALIZER;
    this.lang_type = JSC$JS_ARRAY;
    this.linenum = ln;
    this.items = items;
}

/* Object initializer. */

function JSC$expr_object_initializer (ln, items)
{
    rjs_debug("JSC$expr_object_initializer:");

    this.etype = JSC$EXPR_OBJECT_INITIALIZER;
    this.lang_type = JSC$JS_OBJECT;
    this.linenum = ln;
    this.items = items;
}

/* Null. */

function JSC$expr_null (ln)
{
    rjs_debug("JSC$expr_null:");

    this.etype = JSC$EXPR_NULL;
    this.lang_type = JSC$JS_NULL;
    this.linenum = ln;
}

/* True. */

function JSC$expr_true (ln)
{
    rjs_debug("JSC$expr_true:");

    this.etype = JSC$EXPR_TRUE;
    this.lang_type = JSC$JS_BOOLEAN;
    this.linenum = ln;
}

/* False. */

function JSC$expr_false (ln)
{
    rjs_debug("JSC$expr_false:");

```

```

    this.etype = JSC$EXPR_FALSE;
    this.lang_type = JSC$JS_BOOLEAN;
    this.linenum = ln;
}

/* Multiplicative expr. */

function JSC$expr_multiplicative (ln, type, e1, e2)
{
    rjs_debug("JSC$expr_multiplicative:");

    this.etype = JSC$EXPR_MULTIPLICATIVE;
    this.linenum = ln;
    this.type = type;
    this.e1 = e1;
    this.e2 = e2;
}

/* Additive expr. */

function JSC$expr_additive (ln, type, e1, e2)
{
    rjs_debug("JSC$expr_additive:");

    this.etype = JSC$EXPR_ADDITIVE;
    this.linenum = ln;
    this.type = type;
    this.e1 = e1;
    this.e2 = e2;
    this.constant_folding = JSC$expr_additive_constant_folding;
}

function JSC$expr_additive_constant_folding ()
{
    if (this.e1.constant_folding)
        this.e1 = this.e1.constant_folding ();
    if (this.e2.constant_folding)
        this.e2 = this.e2.constant_folding ();

    /* This could be smarter. */
    if (this.e1.lang_type && this.e2.lang_type
        && this.e1.lang_type == this.e2.lang_type)
    {
        switch (this.e1.lang_type)
        {
            case JSC$JS_INTEGER:
                return new JSC$expr_integer (this.linenum,
                    this.type == '+'
                    ? this.e1.value + this.e2.value
                    : this.e1.value - this.e2.value);

            break;

            case JSC$JS_FLOAT:
                return new JSC$expr_float (this.linenum,
                    this.type == '+'

```



```

                                ? this.e1.value + this.e2.value
                                : this.e1.value - this.e2.value);

        break;

    case JSC$JS_STRING:
        if (this.type == '+')
            /* Only the addition is available for the strings. */
            return new JSC$expr_string (this.linenum,
                                         this.e1.value + this.e2.value);

        break;

    default:
        /* FALLTHROUGH */
        break;
    }

    return this;
}

/* Shift expr. */

function JSC$expr_shift (ln, type, e1, e2)
{
    rjs_debug("JSC$expr_shift:");

    this.etype = JSC$EXPR_SHIFT;
    this.linenum = ln;
    this.type = type;
    this.e1 = e1;
    this.e2 = e2;
}

/* Relational expr. */

function JSC$expr_relational (ln, type, e1, e2)
{
    rjs_debug("JSC$expr_relational:");

    this.etype = JSC$EXPR_RELATIONAL;
    this.lang_type = JSC$JS_BOOLEAN;
    this.linenum = ln;
    this.type = type;
    this.e1 = e1;
    this.e2 = e2;
}

/* Equality expr. */

function JSC$expr_equality (ln, type, e1, e2)
{
    rjs_debug("JSC$expr_equality:");

    this.etype = JSC$EXPR_EQUALITY;
    this.lang_type = JSC$JS_BOOLEAN;
    this.linenum = ln;
    this.type = type;

```

```

    this.e1 = e1;
    this.e2 = e2;
}

/* Bitwise and expr. */

function JSC$expr_bitwise_and (ln, e1, e2)
{
    rjs_debug(" JSC$expr_bitwise_and:");

    this.etype = JSC$EXPR_BITWISE;
    this.linenum = ln;
    this.e1 = e1;
    this.e2 = e2;
}

/* Bitwise or expr. */

function JSC$expr_bitwise_or (ln, e1, e2)
{
    rjs_debug("JSC$expr_bitwise_or:");

    this.etype = JSC$EXPR_BITWISE;
    this.linenum = ln;
    this.e1 = e1;
    this.e2 = e2;
}

/* Bitwise xor expr. */

function JSC$expr_bitwise_xor (ln, e1, e2)
{
    rjs_debug("JSC$expr_bitwise_xor:");

    this.etype = JSC$EXPR_BITWISE;
    this.linenum = ln;
    this.e1 = e1;
    this.e2 = e2;
}

/* Logical and expr. */

function JSC$expr_logical_and (ln, e1, e2)
{
    rjs_debug("JSC$expr_logical_and:");

    this.etype = JSC$EXPR_LOGICAL;

    if (e1.lang_type && e2.lang_type
        && e1.lang_type == JSC$JS_BOOLEAN && e2.lang_type == JSC$JS_BOOLEAN)
        this.lang_type = JSC$JS_BOOLEAN;

    this.linenum = ln;
    this.e1 = e1;
    this.e2 = e2;
}

```

```

/* Logical or expr. */
function JSC$expr_logical_or (ln, e1, e2)
{
  rjs_debug("JSC$expr_logical_or:");

  this.etype = JSC$EXPR_LOGICAL;

  if (e1.lang_type && e2.lang_type
      && e1.lang_type == JSC$JS_BOOLEAN && e2.lang_type == JSC$JS_BOOLEAN)
    this.lang_type = JSC$JS_BOOLEAN;

  this.linenum = ln;
  this.e1 = e1;
  this.e2 = e2;
}

/* New expr. */
function JSC$expr_new (ln, expr, args)
{
  rjs_debug("JSC$expr_new:");

  this.etype = JSC$EXPR_NEW;
  this.linenum = ln;
  this.expr = expr;
  this.args = args;
}

/* Object property expr. */
function JSC$expr_object_property (ln, expr, id)
{
  rjs_debug("JSC$expr_object_property:" + id);

  this.etype = JSC$EXPR_OBJECT_PROPERTY;
  this.linenum = ln;
  this.expr = expr;
  this.id = id;
}

/* Object array expr. */
function JSC$expr_object_array (ln, expr1, expr2)
{
  rjs_debug("JSC$expr_object_array:");

  this.etype = JSC$EXPR_OBJECT_ARRAY;
  this.linenum = ln;
  this.expr1 = expr1;
  this.expr2 = expr2;
}

/* Call. */
function JSC$expr_call (ln, expr, args)
{

```

```

    rjs_debug("JSC$expr_call:");

    this.etype = JSC$EXPR_CALL;
    this.linenum = ln;
    this.expr = expr;
    this.args = args;
}

/* Assignment. */

function JSC$expr_assignment (ln, type, expr1, expr2)
{
    rjs_debug("JSC$expr_assignment:");

    this.etype = JSC$EXPR_ASSIGNMENT;
    this.linenum = ln;
    this.type = type;
    this.expr1 = expr1;
    this.expr2 = expr2;
}

/* Quest colon. */

function JSC$expr_quest_colon (ln, e1, e2, e3)
{
    rjs_debug("JSC$expr_quest_colon:");

    this.etype = JSC$EXPR_QUEST_COLON;
    this.linenum = ln;
    this.e1 = e1;
    this.e2 = e2;
    this.e3 = e3;
}

/* Unary. */

function JSC$expr_unary (ln, type, expr)
{
    rjs_debug("JSC$expr_unary:");

    this.etype = JSC$EXPR_UNARY;
    this.linenum = ln;
    this.type = type;
    this.expr = expr;
}

/* Postfix. */

function JSC$expr_postfix (ln, type, expr)
{
    rjs_debug("JSC$expr_postfix:");

    this.etype = JSC$EXPR_POSTFIX;
    this.linenum = ln;
    this.type = type;
    this.expr = expr;
}

```

/\* Postfix. \*/

```
function JSC$expr_comma (ln, expr1, expr2)
{
  rjs_debug("JSC$expr_comma:");

  this.etype = JSC$EXPR_COMMA;
  this.linenum = ln;
  this.expr1 = expr1;
  this.expr2 = expr2;
}
```

006280"062900

```
/*  
Local variables:  
mode: c  
End:  
*/
```

09650273.082900

00650273 082900

```
/*
 * Input stream definitions.
 * Copyright (c) 1998 New Generation Software (NGS) Oy
 *
 * Author: Markku Rossi <mtr@ngs.fi>
 */

/*
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public
 * License along with this library; if not, write to the Free
 * Software Foundation, Inc., 59 Temple Place - Suite 330, Boston,
 * MA 02111-1307, USA
 */
/*
 * The GNU Library General Public License may also be downloaded at
 * http://www.gnu.org/copyleft/gpl.html.
 */

/*****
 *
 * This software was modified by Yahoo! Inc. under the terms
 * of the GNU Library General Public License(LGPL). For all
 * legal, copyright, and technical issues relating to how
 * this software can be used under GNU LGPL, please write to
 *
 * GNU Compliance, Legal Dept., Yahoo! Inc.,
 * 3420 Central Expressway, Santa Clara, California U.S.A.
 *
 *****/

/*
 * $Source: /usr/local/cvsroot/ngs/js/jsc/streams.js,v $
 * $Id: streams.js,v 1.2 1998/10/26 15:25:21 mtr Exp $
 */

/*
 * File stream.
 */

function JSC$StreamFile (name)
{
    this.name = name;
    this.stream = new File (name);
    this.error = "";

    this.open          = JSC$StreamFile_open;
```

```

    this.close          = JSC$StreamFile_close;
    this.rewind         = JSC$StreamFile_rewind;
    this.readByte       = JSC$StreamFile_read_byte;
    this.ungetByte      = JSC$StreamFile_unget_byte;
    this.readln         = JSC$StreamFile_readln;
}

```

```

function JSC$StreamFile_open ()
{
    if (!this.stream.open ("r"))
    {
        this.error = System.strerror (System.errno);
        return false;
    }

    return true;
}

```

```

function JSC$StreamFile_close ()
{
    return this.stream.close ();
}

```

```

function JSC$StreamFile_rewind ()
{
    return this.stream.setPosition (0);
}

```

```

function JSC$StreamFile_read_byte ()
{
    return this.stream.readByte ();
}

```

```

//@@ function JSC$StreamFile_unget_byte (byte)
function JSC$StreamFile_unget_byte (_byte)
{
    this.stream.ungetByte (_byte);
}

```

```

function JSC$StreamFile_readln ()
{
    return this.stream.readln ();
}

```

```

/*
 * String stream.
 */

```

```

function JSC$StreamString (str)
{

```



```

this.name = "StringStream";
this.string = str;
this.pos = 0;
this.unget_byte = -1;
this.error = "";

this.open          = JSC$StreamString_open;
this.close         = JSC$StreamString_close;
this.rewind        = JSC$StreamString_rewind;
this.readByte      = JSC$StreamString_read_byte;
this.ungetByte     = JSC$StreamString_unget_byte;
this.readln        = JSC$StreamString_readln;
}

function JSC$StreamString_open ()
{
    return true;
}

function JSC$StreamString_close ()
{
    return true;
}

function JSC$StreamString_rewind ()
{
    this.pos = 0;
    this.unget_byte = -1;
    this.error = "";
    return true;
}

function JSC$StreamString_read_byte ()
{
    var ch;

    if (this.unget_byte != "-1")          //@@ if (this.unget_byte >= 0)
    {
        ch = this.unget_byte;
        this.unget_byte = "-1";          //@@ this.unget_byte = -1;
        return ch;
    }

    if (this.pos >= this.string.length)
        return -1;

    //@@ return this.string.charCodeAt (this.pos++);
    return this.string.charAt (this.pos++);
}

//@@ function JSC$StreamString_unget_byte (byte)
function JSC$StreamString_unget_byte (_byte)

```

```
{
  this.unget_byte = _byte;
}
```

```
//@@ NOT used
```

```
function JSC$StreamString_readln ()
{
  var line = new String ("");
  var ch;

  while ((ch = this.readByte ()) != -1 && ch != '\n')
    line.append (String.pack ("C", ch));

  return line;
}
```

006280" 022900

```
/*  
Local variables:  
mode: c  
End:  
*/
```

09650273.082900

09650273 082900

```
/*
 * Internal definitions.
 * Copyright (c) 1998 New Generation Software (NGS) Oy
 *
 * Author: Markku Rossi <mtr@ngs.fi>
 */

/*
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public
 * License along with this library; if not, write to the Free
 * Software Foundation, Inc., 59 Temple Place - Suite 330, Boston,
 * MA 02111-1307, USA
 */
/*
 * The GNU Library General Public License may also be downloaded at
 * http://www.gnu.org/copyleft/gpl.html.
 */

/*****
 *
 * This software was modified by Yahoo! Inc. under the terms
 * of the GNU Library General Public License (LGPL). For all
 * legal, copyright, and technical issues relating to how
 * this software can be used under GNU LGPL, please write to:
 *
 * GNU Compliance, Legal Dept., Yahoo! Inc.,
 * 3420 Central Expressway, Santa Clara, California U.S.A.
 *****/

/*
 * $Source: /usr/local/cvsroot/ngs/js/jsc/compiler.js,v $
 * $Id: compiler.js,v 1.42 1999/01/11 09:01:33 mtr Exp $
 */

/*
 * Constants.
 */

/* Tokens. */

JSC$tEOF      = 128;
JSC$tINTEGER  = 129;
JSC$tFLOAT    = 130;
JSC$tSTRING   = 131;
JSC$tIDENTIFIER = 132;
```

```

JSC$tBREAK = 133;
JSC$tCONTINUE = 134;
JSC$tDELETE = 135;
JSC$tELSE = 136;
JSC$tFOR = 137;
JSC$tFUNCTION = 138;
JSC$tIF = 139;
JSC$tIN = 140;
JSC$tNEW = 141;
JSC$tRETURN = 142;
JSC$tTHIS = 143;
JSC$tTYPEOF = 144;
JSC$tVAR = 145;
JSC$tVOID = 146;
JSC$tWHILE = 147;
JSC$tWITH = 148;

```

```

JSC$tCASE = 149;
JSC$tCATCH = 150;
JSC$tCLASS = 151;
JSC$tCONST = 152;
JSC$tDEBUGGER = 153;
JSC$tDEFAULT = 154;
JSC$tDO = 155;
JSC$tENUM = 156;
JSC$tEXPORT = 157;
JSC$tEXTENDS = 158;
JSC$tFINALLY = 159;
JSC$tIMPORT = 160;
JSC$tSUPER = 161;
JSC$tSWITCH = 162;
JSC$tTHROW = 163;
JSC$tTRY = 164;

```

```

JSC$tNULL = 165;
JSC$tTRUE = 166;
JSC$tFALSE = 167;

```

```

JSC$tEQUAL = 168;
JSC$tNEQUAL = 169;
JSC$tLE = 170;
JSC$tGE = 171;
JSC$tAND = 172;
JSC$tOR = 173;
JSC$tPLUSPLUS = 174;
JSC$tMINUSMINUS = 175;
JSC$tMULA = 176;
JSC$tDIVA = 177;
JSC$tMODA = 178;
JSC$tADDA = 179;
JSC$tSUBA = 180;
JSC$tANDA = 181;
JSC$tXORA = 182;
JSC$tORA = 183;
JSC$tLSIA = 184;
JSC$tLSHIFT = 185;

```

```

JSC$trSHIFT = 186;
JSC$trrSHIFT = 187;
JSC$trSIA = 188;
JSC$trRSA = 189;
JSC$trSEQUAL = 190;
JSC$trSNEQUAL = 191;

```

```
/* Expressions. */
```

```

JSC$EXPR_COMMA = 0;
JSC$EXPR_ASSIGNMENT = 1;
JSC$EXPR_QUEST_COLON = 2;
JSC$EXPR_LOGICAL = 3;
JSC$EXPR_BITWISE = 4;
JSC$EXPR_EQUALITY = 5;
JSC$EXPR_RELATIONAL = 6;
JSC$EXPR_SHIFT = 7;
JSC$EXPR_MULTIPLICATIVE = 8;
JSC$EXPR_ADDITIVE = 9;
JSC$EXPR_THIS = 10;
JSC$EXPR_NULL = 11;
JSC$EXPR_TRUE = 12;
JSC$EXPR_FALSE = 13;
JSC$EXPR_IDENTIFIER = 14;
JSC$EXPR_FLOAT = 15;
JSC$EXPR_INTEGER = 16;
JSC$EXPR_STRING = 17;
JSC$EXPR_CALL = 18;
JSC$EXPR_OBJECT_PROPERTY = 19;
JSC$EXPR_OBJECT_ARRAY = 20;
JSC$EXPR_NEW = 21;
JSC$EXPR_DELETE = 22;
JSC$EXPR_VOID = 23;
JSC$EXPR_TYPEOF = 24;
JSC$EXPR_PREFIX = 25;
JSC$EXPR_POSTFIX = 26;
JSC$EXPR_UNARY = 27;
JSC$EXPR_REGEX = 28;
JSC$EXPR_ARRAY_INITIALIZER = 29;
JSC$EXPR_OBJECT_INITIALIZER = 30;

```

```
/* Statements */
```

```

JSC$STMT_BLOCK = 0;
JSC$STMT_FUNCTION_DECLARATION = 1;
JSC$STMT_VARIABLE = 2;
JSC$STMT_EMPTY = 3;
JSC$STMT_EXPR = 4;
JSC$STMT_IF = 5;
JSC$STMT_WHILE = 6;
JSC$STMT_FOR = 7;
JSC$STMT_FOR_IN = 8;
JSC$STMT_CONTINUE = 9;
JSC$STMT_BREAK = 10;
JSC$STMT_RETURN = 11;
JSC$STMT_WITH = 12;

```

```

JSC$STMT_TRY           = 13;
JSC$STMT_THROW         = 14;
JSC$STMT_DO_WHILE      = 15;
JSC$STMT_SWITCH        = 16;
JSC$STMT_LABELED_STMT  = 17;

```

```
/* JavaScript types. */
```

```

JSC$JS_UNDEFINED = 0;
JSC$JS_NULL      = 1;
JSC$JS_BOOLEAN   = 2;
JSC$JS_INTEGER   = 3;
JSC$JS_STRING    = 4;
JSC$JS_FLOAT     = 5;
JSC$JS_ARRAY     = 6;
JSC$JS_OBJECT    = 7;
JSC$JS_BUILTIN   = 11;

```

```

/*****
 * @@ Token to string
 *****/

```

```
function rjs_t2s(token)
```

```

{
    if (token == JSC$tMULA ) return "*=";
    else if (token == JSC$tDIVA ) return "/=";
    else if (token == JSC$tMODA ) return "%=";
    else if (token == JSC$tADDA ) return "+=";
    else if (token == JSC$tSUBA ) return "-=";
    else if (token == JSC$tLSIA ) return "<<=";
    else if (token == JSC$tRSIA ) return ">>=";
    else if (token == JSC$tRRSA ) return ">>>=";
    else if (token == JSC$tANDA ) return "&=";
    else if (token == JSC$tXORA ) return "^=";
    else if (token == JSC$tORA ) return "|=";
    else if (token == JSC$tEQUAL) return "==";
    else if (token == JSC$tNEQUAL) return "!=";
    else if (token == JSC$tSEQUAL) return "===";
    else if (token == JSC$tSNEQUAL) return "!==";
    else if (token == JSC$tOR) return "||";
    else if (token == JSC$tAND) return "&&";
    else if (token == JSC$tLE) return "<=";
    else if (token == JSC$tGE) return ">=";
    else if (token == JSC$tLSHIFT) return "<<";
    else if (token == JSC$tRSHIFT) return ">>";
    else if (token == JSC$tRRSHIFT) return ">>>";
    else if (token == JSC$tDELETE) return "delete ";
    else if (token == JSC$tVOID) return "void ";
    else if (token == JSC$tTYPEOF) return "typeof ";
    else if (token == JSC$tPLUSPLUS) return "++";
    else if (token == JSC$tMINUSMINUS) return "--";
    else return "" + token;
}

```